Intelligent Speed Adaptation in Curves for Autonomous Vehicles

David Partouche

INTELLIGENT SPEED ADAPTATION IN CURVES FOR AUTONOMOUS VEHICLES

Internship report submitted in partial fulfillment

of the requirements for the degree of Magistere

in Computer Science at the University of Joseph Fourier,Grenoble

September, 2006

David PARTOUCHE

Tutor :

Michel Pasquier

Director, Centre for Computational Intelligence
Associate Professor, Div of Computer Science Nanyang Technological University

Co-tutor :

Anne Spalanzani

Maître de conférences, UPMF, IUT2 Grenoble, Chercheur, Laboratoire GRAVIR, Institut IMAG

Abstract

Millions of road users and pedestrians are killed in traffic accidents each year. The need to increase road safety is one of the major concerns, and the better way to increase safety is to develop systems which are able to automatically drive, the principal cause of road accidents being human error.
Centre for Computational Intelligence (C^2i) has developed a simulated system based on fuzzy neural network, which is able to drive on highway, and to take some decisions like lane changing and car following or overtaking.

The aim of our project is to go one step beyond this system, by implementing intelligent speed adaptation (ISA), so the car can anticipate curves by adapting its speed according to the degree of curvature of the road.
In order to implement ISA, we will use the GenSoYagerFNN, a fuzzy neural network developed in the laboratory, which has shown good performances for autonomous driving.

The information about the environment is given by a camera put in front of the car, and image processing is used for lane detection and to extract necessary data. These data are then fed into the network, which gives in output the vehicle controls (i.e. steering, brake and throttle). The GenSoYagerFNN must first learn from a training set, given by collecting data from a human driver, in order to be able to drive correctly the vehicle.

Experimental results have shown that the GenSoYager was able to correctly adapt its speed according to the curvature of the road, in the same way that human do. The learning process from a human training set has been a success for a simple test, and results are encouraging for the continuation of the project.

Acknowledgments

First of all I would like to thank Anne Spalanzani who introduced me in the world of research, and specially the evolutionary robotics, a really fascinating world. She knows how to encourage and help me when I need it.
I would also like to thank Michel Pasquier for having invited me in his laboratory, and for his advices during this project. Though I took time to understand and begin my project, now the work I do is very interesting, and I hope to keep this way until the end of my internship.
The others "internship" thanks are for Christian Laugier for having let me go to Singapore and *Region Rhone-Alpes* for its financial help.

General thanks go for my parents and family for their help, all the people I met in Singapore (who show me that notion of respect, kindness and responsibility aren't lost! does it only exist in Asia?) and Varsovie, for the music and the real friendship.

At last but not least Sophie, for all (and much more!).

Adaptation de Vitesse dans les Virages pour les Voitures Automatiques - Résumé

Le problème de la sécurité routière prend de plus en plus une place importante dans notre société. Les accidents augmentent d'année en année, le nombre moyen de tués sur la route étant de 12 millions, et le World Health Organization estime une augmantation de ces accidents de 65% d'ici les 20 prochaines années. Ces accidents sont la plupart du temps dûs à une erreur humaine. Pour réduire le nombre d'accidents, il faut donc trouver un moyen de réduire le risque d'erreurs que fait un conducteur sur la route. Des laboratoires de recherches se sont donc focalisés sur la conduite automatique, qui permet de prendre la place du conducteur, et donc d'éviter des accidents dûs à des fautes humaines.

Notre projet s'oriente vers l'anticipation des courbes en conduite automatique, une branche encore non exploitée des *intelligent autonomous vehicles* (IAV). Afin de pouvoir anticiper les courbes, le véhicule doit être capable de les détecter. Pour cela nous utilisons un systeme de vision, via une caméra placée à l'avant du véhicule qui va analyser l'environnement afin de récupérer les données qui nous intéressent. Une fois ces données récupérées, nous allons les traiter par un système hybride, le *GenSoYager Fuzzy Neural Network*, développé par le laboratoire C^2i. Ce système reprend les avantages des systèmes flous et des réseaux de neurones, c'est-à-dire la capacité de raisonner à un niveau d'abstraction élevé (voir sémantique) des systèmes flous, et la capacité d'apprendre et de généraliser à partir d'un ensemble de données pour les réseaux de neurones.

Afin d'effectuer nos expériences, nous avons adapté un simulateur de voitures réaliste, TORCS[1]. Le précédent simulateur développé par le laboratoire C^2i ne gérait pas les forces de Corioli, ce qui nous empêchait de tester notre implémentation de l'adaptation de vitesse pour les courbes dangereuses (en effet, la voiture n'était pas entraînée hors de la route lorsqu'elle arrivait trop vite dans la courbe). Les principales modifications effectuées sur ce simulateur ont été le portage d'un système de vision pour récupérer l'image de la caméra placée sur le véhicule, et de la librairie du rseau de neurones.

[1] http://torcs.sourceforge.net

Afin de conduire le véhicule dans TORCS, nous avons dû prendre le contrôle du volant, de la pédale d'accélération et de la pédale de frein (la boîte de transmission est en mode automatique). La prise de décision sur chacun de ces contrôles est effectuée par un réseau *GenSoYager*. Afin que le réseau puisse efficacement prendre le contrôle du véhicule, nous avons dû l'entraîner à partir de données précédemment enregistrées par un conducteur humain.

Après entraînement du réseau, celui-ci s'avérait capable de correctement conduire la voiture, en adaptant sa vitesse lorsqu'il arrivait dans les courbes, tout comme un être humain. En analysant le réseau ainsi construit, nous avons vu que le GenSoYager générait des règles proches du raisonnement humain, par exemple :

> **SI** la vitesse de la voiture est lente **ET** la route tourne légèrement à droite **ALORS** on accélère normalement

Dans la deuxième étape de notre projet, nous avons implémenté un système d'anticipation, afin de prévoir en avance l'arrivée des courbes. Les expérimentations effectuées sur des circuits tests ont montrées que le vehicule était capable d'anticiper une courbe et donc de ralentir suffisament assez tôt, mais également que la voiture ralentissait suivant le degré et le radius de cette courbe.

Les résultats que nous avons trouvés ont été concluants, mais ne sont qu'une étape du projet. En effet, l'étape finale est de généraliser les résultats, afin que le véhicule soit capable de conduire sur n'importe quelle route après apprentissage sur une route test.

Contents

List of Figures

Chapter 1

Introduction

1.1 Presentation of the Laboratory

The Centre for Computational Intelligence (C^2i) [1], formerly known as the Intelligent Systems Laboratory (iSL), is a new research centre established in April 2004 at the School of Computer Engineering, Nanyang Technological University, Singapore. C^2i comprises faculty staff and students undertaking high quality research in the technologies required for the realization of the intelligent systems of the future.

C^2i's expertise ranges from classical, knowledge-intensive AI, through machine learning and adaptive systems, to nature-inspired Computational Intelligence. C^2i's work focuses on the investigation, design, and development of novel models and techniques and their application for building computing systems and devices that are of practical use in human endeavors.

The centre investigates natural and/or artificial systems to comprehend principles that render intelligent behaviour possible in complex changing environments. C2i's main objective is to devise computing systems that can learn, reason, explain, understand data, and provide solutions to real-world problems. Focus research areas include Adaptive and Autonomous Systems, Neuro-Cognitive Informatics, Decision Support Systems, and Nature-Inspired Systems, while applications target domains such as Automation and Control, Financial Engineering, Humanized Interfaces, Personalized Medicine, and Transportation Systems.

1.2 Background

A world report on road traffic injury prevention has been published by the World Health Organization, estimating 1.2 millions people are killed in road crashes each year, and as many as 50 millions are injured [1]. Projections indicate that these figures will increase about 65% over the next 20 years. The global cost of road crashed and injuries is estimated to be US$ 518 billion per year.

[1] website at http://www.c2i.ntu.edu.sg/

The causes of these accidents are attributed to human error, alcohol, bad weather, heavy traffic or bad infrastructures. Some of these causes can be managed by measures, but the primary cause is human error, and improving the security won't help with this kind of error. Autonomous driving systems have resulted from researches to decrease the risk of human error.

The Intelligent Autonomous Vehicle (IAV) focuses on modeling the human driver in aspects of perception of the environment, learning and reasoning. While several researches have been made and proved effective on highways, with simple scenarios (lane marking, slow changes in curvature, no crossing ...), automated driving in more diverse and complex environment is still far away from being implemented . An average person can easily drive in a city with stops, pedestrian crossings, lane changing, accelerating or slowing down according to the environment changes, but these tasks are hard to reproduce on a computer. Even on Highways Automated Systems, longitudinal control of the vehicle is very simple (like accelerating until the speed limit, braking or decelerating if there's an obstacle in front of the vehicle, accelerating for overtaking...) but more complex behaviors, like anticipating the curves, are not yet implemented.

Intelligent Speed Adaptation is a system which can regulate the vehicle speed on roads [2]. The system has been tested and proved efficient in several countries, but these systems only react to the speed limit of the road, and for the more complex systems, they can also adapt their speed to the road and weather conditions. But none of them can slow down when the car reaches a curve dangerously, which can cause the vehicle to go off the road and create an accident. Though anticipating curves is important for the security of the driver, any article has yet been published on this subject, this topic being quite innovative in the ISA research.

Driving can be modelled as a continuous decision-making process involving a set of rules that relate sensory input to control output. But designing these rules is quite difficult, so the simplest way is to learn from human expertise for extracting the rules. Our approach uses a type of hybrid intelligent system: the *Fuzzy Neural Network*, developped by C^2i. A fuzzy neural network is a combination of a neural network and a fuzzy system, which provides advantage of both: the learning and generalization of neural networks, and the reasoning strength and ease of interpretation of fuzzy systems. A simulator, also developped by C^2i, has been used to test the capacity of the GenSoYager to learn how to drive from a human. Successful manoeuvres achieved so far include reverse parking, U-turn, as well as highway driving.

This project focuses on the use of GenSoYagerFNN, a novel fuzzy neural network, which is already working for highways driving and some tactical maneuvers, like lane changing, overtaking, car following, and collision avoidance [3] [4] [5].

1.3 Objectives

The aim of our project is to anticipate curves, by adapting the speed of our automated vehicle to the degree of the road's curvature. In order to adapt its speed, the vehicle must be able to see the curve (via a camera vision system) and to roughly calculate the degree of this one.

So the first goal is to find a way to compute the degree of a road curve given a picture of this track. To do so, we must first detect the current lane the car is on. This is done by a technique of image processing called *lane tracking*. The image processing must be able to give the position of the car, relatively to its current lane, and to see the the road the farthest away in order to be able to anticipating the curve.

Then, once data are extracted from the image, these one are used as inputs to the GenSoYagerFNN. Finding the right data that must be sent to the network is very experimental, and a lot of tests have been done to find which one gives the best results. We use three networks for our system, one is for the lateral control (i.e. steering) and the two others for longitudinal control (i.e. throttle and brake). The combination of these two networks allow the system to drive the vehicle.

Experimenting using the previously released car simulator (Fuzzylot) was not possible, due to a problem in modeling the car physics, namely the lack of adequate lateral acceleration when the vehicle high speed drove in curves. We decided to use another simulator, TORCS, which is freely available with its sources, and under GPL licence. So we had to map the TORCS driving system with the artificial intelligence of the last simulator, to automatically control TORCS's cars. This was done by adapting the vision system and the GenSoYagerFNN of Fuzzylot to TORCS, but some problems were encountered due to the passage of windows (for Fuzzylot) to Linux programming environment.

1.4 Scope

The scope of this project comprises of the following:

- To have an overview of the GenSoYagerFNN and the past projects involved in the car simulator
- adapting the TORCS simulator for our needs, and mapping it with the image processing and the GenSoYager, to get intelligent automated vehicles in the simulator
- Testing the system on a simple track (without big curves, as the GenSoYager is not yet able to slow down given a curve)
- Implementing the speed adaptation, and experimenting with different types of curves
- Adding anticipation to the longitudinal control

1.5 Report Organization

The report is organized as follow. The first chapter gives a overview of the state of art in automated intelligent vehicles, and states the objectives. Chapter 2 gives a brief discussion about literature review on the previous related work, the fuzzy neural networks, and especially the GenSoYagerFNN. Chapter 3 gives an overview of the simulator software used in the project. Detailed description about design and implementation of Intelligent Speed Adaption on the TORCS simulator can be found in chapter 4. Chapter 5 discusses about the experimental results and analysis in the investigation of the realized speed adaptation. Finally, Chapter 6 provides the summary of the project, including accomplishment and the future work. A list of references is also included in annex.

Chapter 2

Literature Review

2.1 Related Work on Autonomous Driving

Autonomous driving is a branch of artificial intelligence, maybe one of the most promising, because of its different fields, like preventing accidents, improving transport efficiency, assisting military operations, even, in a more cognitive field, helping for understanding the human brain.

Autonomous driving can have different approaches, classified in three main areas:

1. collision warning systems (like rear impact warning, pedestrian detection, collision warning ...)

2. driver assisting systems (like adaptive cruise control, lane keeping, maneuvering ...)

3. vehicle automation

We will focus on vehicle automation, describing different systems such as ALVINN, BATMobile and SAPIENT.

2.1.1 ALVINN : Autonomous Land Vehicle In a Neural Network

ALVINN is a 3-layer back propagation network designed for the task of road following [6]. ALVINN takes images from a camera and a laser range finder as input, and produces as output the direction the vehicle should travel in order to follow the road (see figure 2.1). Network training is performed using 1200 artificial road snapshots. Back-propagation is conducted using this set of examples. After 40 epochs of training, the network correctly dictates a correct turn curvature approximately 90% of the time.
Tests have been conducted on the NAVLAB, a modified van (equipped with sensors and material for calculation). The network could drive the NAVLAB along a 400 meter path.

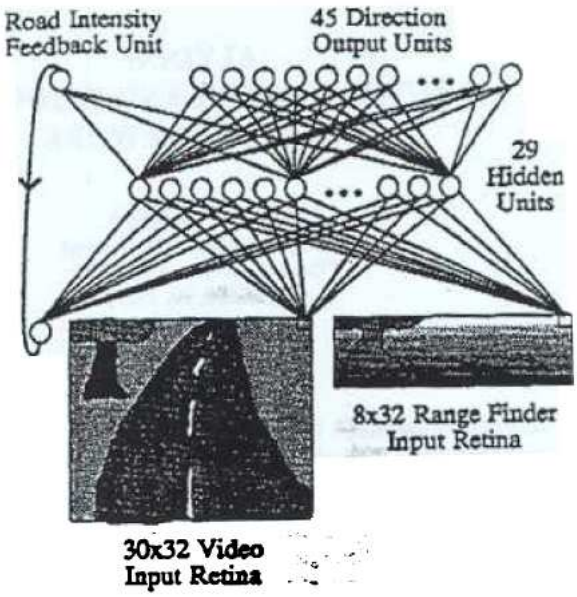

Figure 2.1: The ALVINN architecture [6]

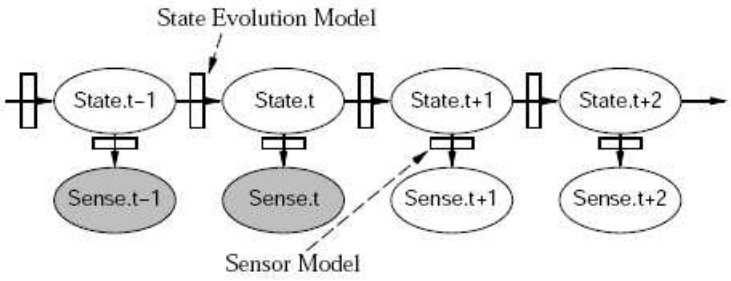

Figure 2.2: The structure of a dynamic probabilistic network [7]

2.1.2 BATMobile: Toward a Bayesian Automated Taxi

The BATMobile is a part of the BAT (Bayesian Automated Taxi) project [7]. The approach is based upon a decision-theoric architecture using dynamic probabilistic networks (DPN). The use of probabilistic networks provide a solution to the problem of sensor noise/failure, and uncertainty about the behavior of other vehicles and about the effects of ones own actions.
Probabilistic networks are directed acyclic graphs in which nodes represent random variables and arcs represent causal connections among the variables. A conditional probability table is associated with each node, and provides the node's possible states given each possible state of its parents.
DPNs allow for reasoning in domains where variables take on different values over time (see figure 2.2).

The network has been tested on several sample traffic scenarios. For each

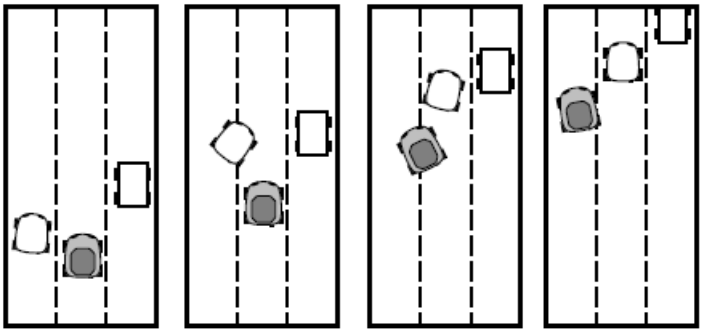

Figure 2.3: Reacting to unsafe drivers [7]

test, the simulator reads a scenario description file, and determines the trajectories of all the vehicles at each simulator "clock tick": it passes current state information in the form of sensor readings to each vehicle controller, which in turn outputs its decision for the current time step.
The goal of the BAT controller is to maintain a target speed in a target line. When other vehicles interfere, the controller makes appropriate acceleration / deceleration and lane-changing maneuvers. Figure 2.3 shows a reaction to an unsafe driver.

2.1.3 SAPIENT: Situation Awareness for Driving in Traffic

SAPIENT(Situation Awareness Planner Implementing Effective Navigation in Traffic) is a reasoning system that combines high-level task goals with low-level sensor constraints to control simulated and real vehicles [8] [9].
SAPIENT consists of a number of reasoning modules whose outputs are combined using a voting scheme. The behavior of these modules is directly dependent on a large number of parameters both internal and external to the modules. Evolutionary algorithms are used to automatically set each module's parameters, avoiding manually selection and providing a easy way to find good parameter settings.

SAPIENT partitions the driving task into many different aspects, each one represented by an independent agent known as a *reasoning object*.
A reasoning object represents a physical entity relevant to the driving task (car ahead, upcoming exit), and every reasoning object takes input from one or more sensors.
All reasoning objects can vote over a predetermined set of actions. Actions have a *longitudinal* and a *lateral* component. Longitudinal commands correspond to speeding up and braking, while lateral commands map to lane changes.
Reasoning objects indicate their preference for a given action by assigning a vote to this action. The magnitude of the vote corresponds to the intensity of the preference and its sign indicates approval or disapproval. Each object must vote to every action (represented by a 3 x 3 matrix, see table 2.1).

Table 2.1: action matrix [8]

decel-left	nil-left	accel-left
decel-nil	nil-nil	accel-nil
decel-right	nil-right	accel-right

SAPIENT uses a voting arbiter which multiply the votes in each reasoning object by a scalar weight, and then chooses the highest cumulative vote for execution in the next time-step.

2.1.4 Comparison of Related Work

The different projects described in this section permit to control automated vehicles, with a different degree of effectiveness (ALVINN can drive in simple scenarios, while BATMobile or SAPIENT can change line to takeover a vehicle). But none of them is able to adapt its speed with regard to the degree of curvature of the road.
To resolve this problem, we will be using the GenSoYagerFNN, a neural fuzzy network, that is already able to drive vehicles on highways, with different tactical scenarios.

But let first describe what neuro-fuzzy systems are, and the principle of the GenSoYagerFNN.

2.2 Neuro-Fuzzy Systems

2.2.1 Fuzzy Systems

Introduction to Fuzzy Systems

Fuzzy systems have been introduced in the earlier 1970's, and are now commonly use in systems that can't be described with a mathematical model, but rather with linguistic terms [10].
Fuzzy systems can be classified as intelligent control, as they incorporate human knowledge into their components (fuzzy sets, fuzzy logic and fuzzy rule bases). They supply systems which can deal with the complex, dynamic and uncertain real world.
Compared with various conventional approaches, fuzzy systems utilize more information from domain experts and yet rely less on mathematical modeling about a physical system.

We will first study the Fuzzy set theory, introduced by Zadeh in 1965 [11].

Fuzzy Set Theory

The classical set theory is built on the fundamental concept of set. An individual is either a member or not a member of a specified set in question, there

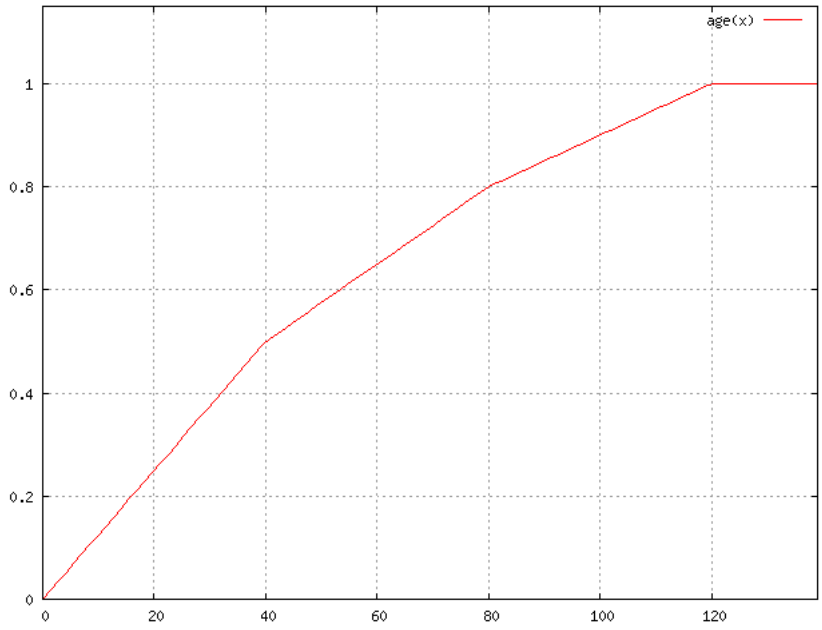

Figure 2.4: A fuzzy membership function for oldness

is a very precise and clear boundary indicating whether or not an individual belongs to the set. In the classical set theory, it is not permissible for an individual to be partially in a set and also partially not in the same set at the same time. On the contrary, fuzzy set theory accepts partial memberships [11]; therefore, in a sense it generalizes the classical set theory to some extent.

Let consider the subset of old people in the set of all human beings. This subset is a fuzzy subset because we can't precisely define the notion of oldness. In classical set theory, one may draw a line at the age of 40. That means a person who is 40 years old is considered old, but a 39 years old person doesn't. This distinction is mathematically correct, but practically unreasonable.
One would better like to describe the concept of oldness by a curve (see figure 2.4), which is more common sense. A 40 years old person is considered to be old with degree of 0.5 and at the same time also young with a degree of 0.5. The curve in figure 2.4 is called a *membership function*.

Consider the subset :

$$S_f = \{\ s \in S \mid s\ is\ old\ \}$$

Suppose that a membership function is associated with it, then this subset along with the chosen membership function (denoted by $\mu_{s_f}(s)$) is called a *fuzzy set*. It is now clear that a fuzzy set consists of two components: a regular set and a membership function associated with it.

Operations on fuzzy sets

α-cut The subset S_α of S_f defined by

$$S_\alpha = \{\ s \in S_f \mid \mu_{S_f}(s) \geq \alpha\ \}$$

is called an *α-cut* of the fuzzy set S_f.

general rule Let X and Y be two fuzzy sets, and consider a two variable function (e.g., $+$, $-$, $\cdot$, $/$, max, min) : $F : X \times Y \to Z$

$$\mu_Z(z) = \bigvee_{z=F(x,y)} \{\ \mu_X(x) \wedge \mu_Y(y)\ \}$$

where, for two real numbers s_1 and s_2, $s_1 \vee s_2 = min\{s_1, s_2\}$ and $s_1 \wedge s_2 = max\{s_1, s_2\}$.
Using the α-cut notation,this is equivalent to the following:

$$\begin{aligned}(Z)_\alpha &= F((X)_\alpha, (Y)_\alpha) \\ &= \{\ z \in Z \mid z = F(x,y), x \in (X)_\alpha, y \in (Y)_\alpha\ \}\end{aligned}$$

Fuzzy Logic Theory

Classical logic deals with propositions that are either *true* or *false*, but not between, nor are both simultaneous. The main content of the classical logic is the study of *rules*. A rule usually takes the following form:

IF x_1 is true AND x_2 is false AND ... AND x_n is false THEN y is false.

The ultimate goal of fuzzy logic theory is to provide a foundation for approximate reasoning using imprecise propositions based on fuzzy set theory.
Consider the following typical example of non-precise reasoning in linguistic terms that cannot be handled by the classical reasoning:

1. Everyone who is 40 to 70 years old is old but is very old if he/she is 71 years old or above; everyone who is 20 to 39 old is young but is very young if he/she is 19 years old or below.

2. David is 40 years old and Mary is 39 years old.

3. David is old but not very old. Mary is young but not very young.

This example is called *approximative reasoning*.
Fuzzy logic can deal with such imprecise inference, allowing the imprecise linguistic terms such as:

- fuzzy predicates: old, rare, severe, expensive, high, fast
- fuzzy quantifiers: many, few, usually, almost, little, much
- fuzzy truth values: very true, true, unlikely true, mostly false, false, definitively false

Let A be a fuzzy set, for any $a, b \in A$, the logical operations are defined as follow:

$$\begin{aligned}
\mu_A(a \wedge b) &:= min\{\ \mu_A(a), \mu_A(b)\ \} = \mu_A(a) \wedge \mu_A(b) \\
\mu_A(a \vee b) &:= max\{\ \mu_A(a), \mu_A(b)\ \} = \mu_A(a) \vee \mu_A(b) \\
\mu_A(\overline{a}) &:= 1 - \mu_A(a) \\
\mu_A(a \Rightarrow b) &:= min\{\ 1, 1 + \mu_A(b) - \mu_A(a)\ \} = \mu_A(a) \Rightarrow \mu_A(b) \\
\mu_A(a \Leftrightarrow b) &:= 1 - |\ \mu_A(a) - \mu_A(b)\ | = \mu_A(a) \Leftrightarrow \mu_A(b)
\end{aligned}$$

Fuzzy Rule Base

Fuzzy rule base is the core of a fuzzy system.
A *general fuzzy IF-THEN rule* has the form:

"IF a_1 is A_1 AND ... AND a_n is A_n THEN b is B"

Using the fuzzy logic AND operation, this rule is implemented by the following evaluation formula:

$$\mu_{A_1}(a_1) \wedge \cdots \wedge \mu_{A_n}(a_n) \Rightarrow \mu_B(b)$$

A *fuzzy rule base* is a set of one or more IF-THEN rules.

Example for a mathematical model

Let $y = f(x)$ be an invertible function defined on $X = [0, 4]$ with a value range $Y = [-4, 0]$ (see figure 2.5). Suppose that the designer doesn't know the actual formula of f.
Let μ_S, μ_M, μ_L be membership functions describing small, medium, and large absolute values in the sets X and Y (see figure 2.5).
Thus, one may approximate the real function $y = f(x)$ by the following fuzzy rule base:

1. "IF x is positive small THEN y is negative small"
2. "IF x is positive medium THEN y is negative medium"
3. "IF x is positive large THEN y is negative large"

2.2.2 Neural Networks

Introduction to artificial neural networks

Artificial neural networks are systems constructed to make use of some organizational principles resembling those of the human brain [12]. They are good at tasks such as pattern classification, function approximation, optimization, vector quantization and data clustering.
Neural networks have a large number of highly interconnected nodes (called *neurons*), that operate in parallel.
The first computational model for an artificial neuron was proposed by McCulloch and Pitts in 1943, and is called a *M-P neuron* [13]. In this model (figure 2.6), the i^{th} processing element computes a weighted sum of its inputs,

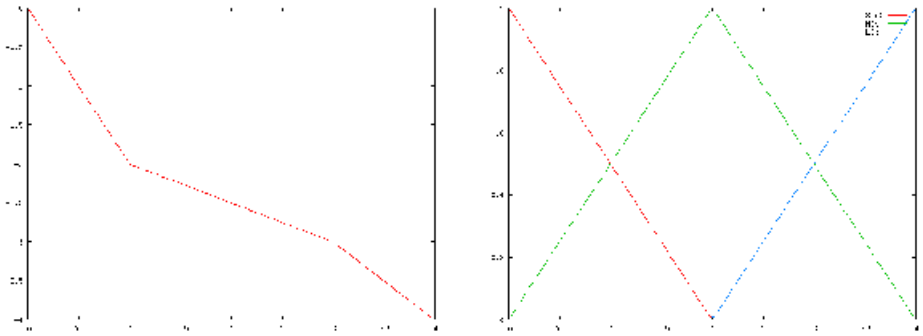

Figure 2.5: An unknown function, and its membership functions

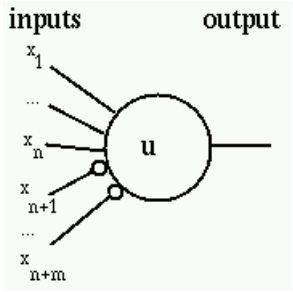

Figure 2.6: schematic diagram of the M-P neuron [a]

[a] http://diwww.epfl.ch/mantra/tutorial/english/mcpits/html/

and outputs $y_i = 1$ (firing) or 0 (not firing) according whether this weighted input sum is above or below a certain threshold θ_i :

$$y_i(t+1) = a(\sum_{j=1}^{m} w_{ij} \cdot x_i(t) - \theta_i)$$

where the activation function is a unit step function:

$$a(f) = \begin{cases} 1 & \text{if } f \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

Feed-forward networks

Let first discussed about the single-layer feed-forward networks, also called *perceptrons* [14].
The perceptron (figure 2.7) has one layer of input nodes, and one layer of output nodes. Each layer is fully connected between the other, but no connections exist between nodes in the same layer. When the input layer sends a signal to the output layer, the associated weights on the connections are applied and each receiving node sums up the incoming values. If the sum exceeds a given threshold, that node in turn fires an output signal.
By adjusting the weights on the connections between layers, the perceptron's

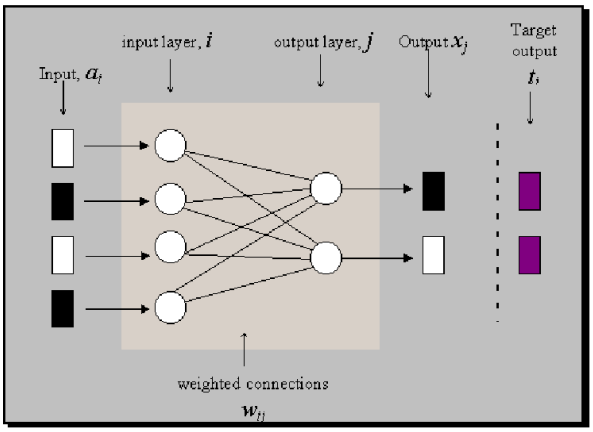

Figure 2.7: A simple perceptron diagram [a]

[a] http://ei.cs.vt.edu/ history/Perceptrons.Estebon.html

output could be "trained" to match a desired output. Training is accomplished by sending a given set of inputs through the network and comparing the results with a set of target outputs. If there is a difference between the actual and the target outputs, the weights are adjusted on the adaptive layer to produce a set of outputs closer to the target values.

The perceptrons can only resolve problems where the input patterns are linearly separable or linearly independents. These limitations don't apply anymore if the network has a *hidden layer*, i.e. a layer between the input and output layers.
The multilayer feed-forward network also has the ability to approximate any non-linear continuous function [15].
One of the most used algorithms in order to learn the weights of the neural network is the back-propagation algorithm. The back-propagation is a supervised learning technique. Given a set of input-output pairs, the algorithm provides a procedure for changing the weights int the network to classify the input patterns correctly. The basis for this weight update algorithm is the gradient-descent method: the input pattern is propagated from the input layer to the output layer, and the result is compared with the desired result. Then the error (difference between the real and desired results) is back-propagated from the output layer to the previous layers for them to update their weights.

2.2.3 Fuzzy Neural Networks

Neural networks are low-level computational structures, while fuzzy systems can reason on higher level. However, since fuzzy system don't have learning capability, it is difficult to tune the fuzzy rules and membership functions from the training data set. Also, because the internal layers of a neural network are very opaque for the user, the mapping rules in the network are not visible and are difficult to understand. Thus, the neuro-fuzzy integration reaps the the benefits of

both neural networks and fuzzy logic systems [16]. The neural networks provide connectionist structure and learning abilities to the fuzzy logic systems, and the fuzzy logic systems provide the neural networks with a structural framework with high-level fuzzy IF-THEN rule thinking and reasoning.
We can characterize the fusion of these two systems in three categories:

Neural Fuzzy Systems the use of neural networks as tools in fuzzy systems

Fuzzy Neural Networks fuzzification of conventional neural networks models

Fuzzy-Neural Hybrid Systems incorporating fuzzy technologies and neural networks into hybrid systems

We will focus our discussion on *Fuzzy Neural Networks*. They retain the basic properties and architectures of neural networks and simply fuzzify some of their elements. A crisp neuron can become fuzzy and the response of the neuron to its lower-layer activation signal can be of a fuzzy relation type rather than a sigmoid type. The synaptic weights can also be replaced by fuzzy parameters.
A lot of different fuzzy neural can be found, like the *fuzzy perceptron* [17], the *fuzzy associative memory* (FAM) [18], the *Fuzzy Kohonen Clustering Network* [19], the *Fuzzy Cerebral Model Articulation Controller* [20] or the *Pseudo Outer-Product Based Fuzzy Neural Network* (POPFNN) [21] [22] among others.

We will now study the framework of the fuzzy neural networks via a specific one: the *GenSoYagerFNN*.

2.3 GenSoYagerFNN

2.3.1 Generic Self-Organizing Fuzzy Neural Network

The Generic Self-organizing Fuzzy Neural Network (GenSoFNN) is a fuzzy neural network with a generic connectionist structure [23].
Most of existing neural fuzzy systems encounter the following problems, they are:

1. *inconsistent* rule-base;
2. heuristically defined node operations;
3. susceptibility to noisy training data and the *stability-plasticity* dilemma;
4. needs for *prior* knowledge such as the number of clusters to be computed.

The GenSoFNN is a new system immune to these deficiencies, and employs a new clustering technique known as discrete incremental clustering (DIC), to enhance its noise tolerance capability.

The training phase of the GenSoFNN (figure 2.8) consists of three phases : *self-organizing, rule formulation and parameter learning*, which will be describe later in this section.

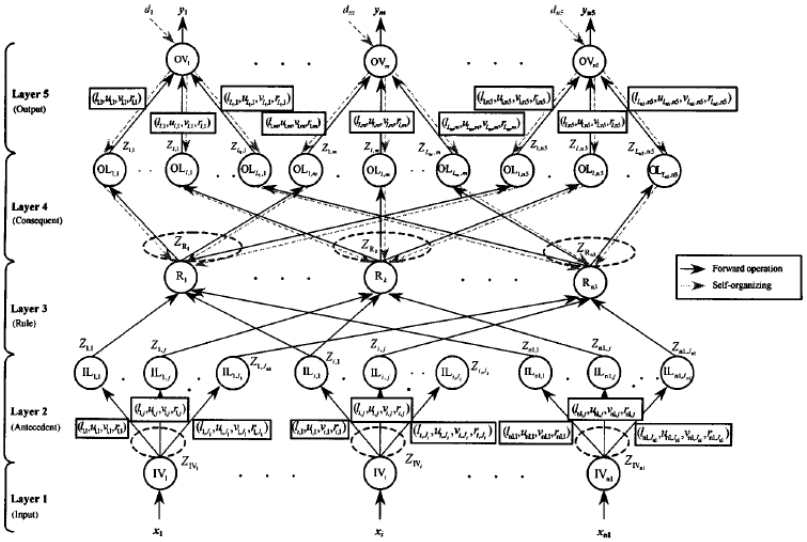

Figure 2.8: Structure of the GenSoFNN [23]

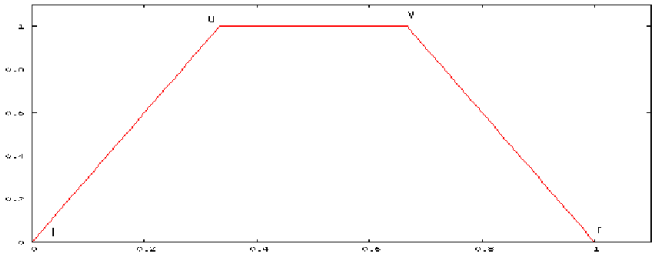

Figure 2.9: Trapezoid-shaped fuzzy set

Structure of the GenSoFNN

The GenSoFNN consists of five layers of nodes. Each input node has a single input, the vector $X = [x_1, \ldots, x_i, \ldots, x_{n1}]$ represents the inputs of the GenSoFNN, and each output node computes a single output, the vector $Y = [y_1, \ldots, y_m, \ldots, y_{n5}]$ denotes the outputs of the GenSoFNN with respect to the X input. In addition, the vector $D = [d_1, \ldots, d_m, \ldots, d_{n5}]$ represents the desired network outputs required during the parameter learning phase of the training cycle.
The trainable weights of the GenSoFNN are found in layers 2 and 5, layer 2 contain the parameter of input fuzzy sets, and layer 5 of output fuzzy sets. These parameters are interpreted as corners of the trapezoid-shaped fuzzy sets computed by the GenSoFNN. They are denoted l and r for left and right support points, and u and v for left and right kernel points (figure 2.9).

The GenSoFNN adopts the Mamdani's fuzzy model [24], and the k^{th} fuzzy rule R_k has the form:

$$R_k \;:\; IF\ x_1\ is\ IL_{(1,j)_k}\ \ldots\ and\ x_i\ is\ IL_{(i,j)_k}\ \ldots\ and\ x_{n1}\ is\ IL_{(n1,j)_k}$$
$$THEN\ y_1\ is\ OL_{(l,1)_k}\ \ldots\ and\ y_m\ is\ OL_{(l,m)_k}\ \ldots\ and\ y_{n5}\ is\ OL_{(l,n5)_k}$$

where $IL_{(i,j)_k}$ is the j^{th} fuzzy label of the i^{th} input that is connected to R_k, and $OL_{(l,m)_k}$ is the l^{th} fuzzy label of the m^{th} output to which R_k is connected.

Self-Organization of GenSoFNN

GenSoFNN models a problem domain by first performing an analysis of the numerical training data, and then deriving the fuzzy rule base from the computed clusters.
The clustering technique used here is the discrete incremental clustering (DIC) technique. This technique has several advantages compared to the hierarchical-based and partition-based techniques. Compared to hierarchical-based techniques the DIC is dynamic, and so can self-organize and self-adapt with changing environments. Then, compared to partition-based techniques, DIC doesn't need prior knowledge, such as the number of classes in the training data. DIC also doesn't suffer from the *stability-plasticity* dilemma where new information cannot be learned without running the risk of eroding old but valid knowledge. In the GenSoFNN, DIC computes trapezoid-shaped fuzzy sets. The DIC technique maintains a consistent representation of the fuzzy sets by performing clustering on a local basis. The proposed DIC has 5 parameters:

1. *Fuzzy Set Support Parameter SLOPE:* Each new cluster begins as a triangular fuzzy set, and its support (l,r segment) is defined by the parameter SLOPE.

2. *Plasticity Parameter β:* a cluster grows by expanding its kernel. This expansion is controlled by the plasticity parameter β. If a cluster is the best-fit for a point, and the point is not included in its kernel, the cluster expends its kernel by an amount of β. To satisfy the *stability-plasticity* dilemma, the initial value of β is preset to 0.5, and decreases as the cluster expanding its kernel.

3. *Tendency Parameter TD:* the tendency parameter represents the cluster's willingness to grow when it is the best-fit to a data point that falls outside its kernel. The value of TD for a newly created cluster is 0.5, and the cluster stops expanding its kernel when TD reaches 0. The rate of decrease depends of the membership value of the data point, and is calculated as follow:
$$TD_{new} = TD_{old} + (-0.5 - TD_{old}) \times (1 - \mu(x))^2$$
μ being the membership function for the cluster, and x the data point.

4. *the input and output thresholds, IT and OT:* the thresholds specify the minimum membership value an data point must have before it is considered as relevant to any existing clusters. if the membership value falls below the threshold, a new cluster is created, based on that data point.

Now let have a look to the DIC algorithm:

```
Assume data set X = X(1), ..., X(P) where P is the number
    of training vectors.
Vector X(p) = x_1^p, ..., x_n1^p represents the p^th input training
    vector to the GenSoFNN.
Initialize STEP and SLOPE ∈ (0, 0.5], IT = β = TD = 0.5
For all training vector p ∈ 1...P do {
  For all input dimensions i ∈ 1...n_1 do {
    if there are no fuzzy labels in the i^th input dimension
      Create a new cluster using x_i^(p)
    else do {
       Winner = arg max_{j∈{i...j}} {μ_{i,j}(x_i^p)}
      if membership value greater than input threshold
        update kernel of Winner
      else
        create new cluster
    }
  }
}
```

Rule Formulation of GenSoFNN

The fuzzy rules are formulated using the RuleMAP rule mapping process. During the rule mapping process, each rule activates is ISP (input space partition, the layer 2 nodes) and OSP (output space partition, the layer 4 nodes), by firing of layers 1 and 2 with a stimulus X feeding into layer 1 for the ISPs, and firing layer 4 and 5 with the desired output D feeding backward from layer 5 (See figure 2.10 for the flowchart of the rule mapping process).
The function *EstLink* identifies the proper connections between the input fuzzy labels (layer 2), the fuzzy rules (layer 3), and the output fuzzy labels (layer 4). More details on the RuleMAP is described in [25].

2.3.2 The Yager Inference Scheme

The original fuzzy inference scheme extended the conventional *modus ponens* rule which states that from the propositions:

P_1: *IF X is A THEN Y is B*
P_2: *X is A*

it can be deduced that *Y is B*.
The proposition P_1 concerns the joint fuzzy variable (X,Y) and is characterized by a fuzzy set over the cross product space $U \times V$. Specifically, P_1 is characterized by a possibly distribution:

$$\Pi_{(X|Y)} = R$$

There are two approaches to interpret the fuzzy relation R. One is based on the *conjunctive* model of fuzzy relation and the other one is based on the *implication-based* model of fuzzy relation.
The *Compositional Rule of Inference* (CRI) scheme adopts the first approach [26]. This is illustrated below:

$$\mu_R(x, y) = min\{\mu_A(x), \mu_B(y)\}$$

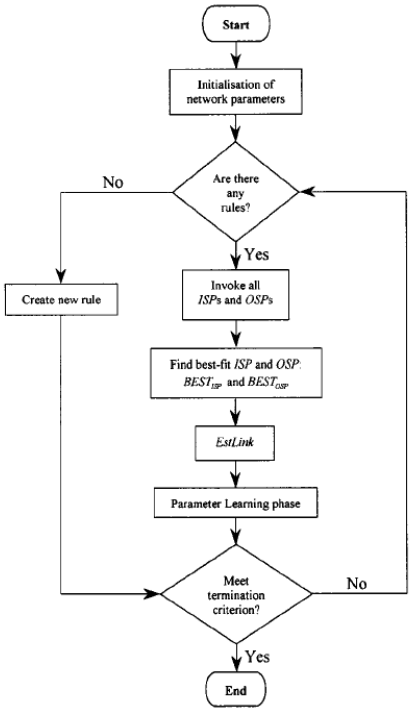

Figure 2.10: Flowchart of RuleMAP [23]

On the other hand, the *Yager Rule of Inference* adopts the second approach, which is based on implication model of fuzzy relation [27]:

$$\mu_R(x, y) = max\{(a - \mu_A(x)), \mu_B(y)\}$$

It should be noted that both the formulas correspond to the logical transition of P_1 interpreted in different ways. The second formula corresponds to the statement $\neg A \cup B$, which is essentially the same as $A \rightarrow B$ in crisp logic. The implication based model of fuzzy relation is exactly the core concept that underpins the *Yager Inference Scheme*.

2.3.3 Mapping of Yager Inference Scheme into the GenSoFNN

A new architecture has been developed by mapping the Yager inference scheme into the GenSoFNN. This new structure is known as the *GenSoYagerFNN*. Now let have a look to the GenSoYager operations in its different layers.

Layer 1: Fuzzyfication

This layer consists of input nodes, which fuzzify the crisp inputs the network receive. For crisp input x_i, it is fuzzified into its corresponding fuzzy set $\overline{X_i}$ using the singleton fuzzifier:

$$\mu_{\overline{X_i}}(X_i) = \begin{cases} 1 & if \overline{X_i} = X_i \\ 0 & \text{otherwise} \end{cases}$$

The operation of singleton fuzzifier is mapped into the input layer:

$$\begin{array}{lrcccc} \text{Net synaptic input of node } IV_i, & Net_{IV_i} & = & f^{(1)}(x_i) & = & X_i \\ \text{Net synaptic output of node } IV_i, & Z_{IV_I} & = & a^{(1)}(Net_i) & = & \overline{X_i} \end{array}$$

Layer 2: Antecedent Matching

The fuzzified inputs from layer 1 are then compared against their corresponding input labels that form the antecedent section of the fuzzy rules in the GenSoYager. The antecedent matching between the inputs and the antecedent section is essentially to compute the negation of membership values of the inputs with respect to the input fuzzy sets.

$$\begin{array}{lrcccc} \text{Net synaptic input of node } IL_{i,j}, & Net_{i,j} & = & f^{(2)}(Z_{IV_i}) & = & \overline{X_i} \\ \text{Net synaptic output of node } IL_{i,j}, & Z_{i,j} & = & a^{(2)}(Net_{i,j}) & = & 1-\mu_{IL_{i,j}}(\overline{X_i}) \\ & & & & & 1-\mu_{IL_{i,j}}(X_i) \end{array}$$

where $\mu_{IL_{i,j}}(X_i)$ is the membership function of input term $IL_{i,j}$.

Layer 3: Rule Fulfillment

The third layer of the GenSoYagerFNN contains the furry rule base of the network. Each rule node R_k computes the degree of fulfillment (i.e. the overall similarity) of the current inputs with respect to the antecedents of the fuzzy rules it denotes. In a fuzzy relation, the antecedent sections of a fuzzy rule R_k are connected by "AND" conjunctive and therefore operator min is used to compute the aggregated rule fulfillment of R_k.

$$\begin{array}{lrcl} \text{Net synaptic input of node } R_k, & Net_{R_k} & = & f^{(3)}(Z_{(1,j)_k}, \ldots, Z_{(n_1,j)_k}) \\ & & = & \{Z_{(1,j)_k}, \ldots, Z_{(n_1,j)_k}\} \\ \text{Net synaptic output of node } R_k, & Z_{R_k} & = & a^{(3)}(Net_{R_k}) \\ & & = & 1 - max\{Z_{(1,j)_k}, \ldots, Z_{(n_1,j)_k}\} \end{array}$$

where n_1 is the number of inputs of the GenSoYagerFNN.

Layer 4: Consequent Derivation

Layer 4 contains output term nodes that represent the output fuzzy sets of the consequent of the rules in layer 3. Each output term may be connected to multiple fuzzy rules indicating that they may have the same consequent. GenSoYagerFNN uses implication-based model of fuzzy relation therefore conclusions of parallel rules will have to be combined *conjunctively*. This is map onto the network in the following form:

$$\begin{array}{lrcl} \text{Net synaptic input of node } OL_{l,m}, & Net_{l,m} & = & f^{(4)}(Z_{R_1}^{(l,m)}, \ldots, Z_{R_{n_3}}^{(l,m)}) \\ & & = & \{Z_{R_1}^{(l,m)}, \ldots, Z_{R_{n_3}}^{(l,m)}\} \\ \text{Net synaptic output of node } OL_{l,m}, & Z_{l,m} & = & a^{(4)}(Net_{l,m}) \\ & & = & 1 - max\{Z_{R_1}^{(l,m)}, \ldots, Z_{R_{n_3}}^{(l,m)}\} \end{array}$$

Layer 5: Output defuzzification

The output nodes are responsible for the defuzzification of the derived fuzzy outputs from the GenSoYagerFNN before presenting them as crisp outputs. For each output y_m, the derived fuzzy conclusion for all its output labels are aggregated using a modified *center of averaging* (COA) technique to produce the final output. This technique is applied to the GenSoYagerFNN as follow:

$$\begin{array}{lrcl} \text{Net synaptic input of node } OV_m, & Net_m & = & f^{(5)}(Z_{(1,m)}, \ldots, Z_{(l_m,m)}) \\ & & = & \{Z_{(1,m)}, \ldots, Z_{(l_m,m)}\} \\ \text{Net synaptic output of node } OV_{l,m}, & y_m & = & a^{(5)}(Net_m) \\ & & = & \frac{\sum_{l=1}^{L_m}((1-Z_{l,m})\times \tilde{\mathrm{m}}_{(l,m)})}{\sum_{l=1}^{L_m}(1-Z_{l,m})} \\ & & = & \frac{MIZSum_m}{IZSum_m} \end{array}$$

where L_m is the number of output term neurons that OV_m has, and $\tilde{\mathrm{m}}_{(l,m)}$ is the mean (i.e. the center) point of the kernel of the fuzzy set represented by $OL_{l,m}$. The parameter $\tilde{\mathrm{m}}_{(l,m)}$ is defined by:

$$\tilde{\mathrm{m}}_{(l,m)} = \frac{U_{(l,m)} + V_{(l,m)}}{2}$$

$U_{(l,m)}$ and $V_{(l,m)}$ are the left and right kernel points of fuzzy set represented by $OL_{(l,m)}$

2.3.4 GenSoYager Parameter Learning

GenSoFNN employs the popular *negative gradient descent back-propagation* algorithm in order to tune the parameters of its fuzzy sets in label layers [28]. The main objective is to minimize the cost function error defined by:

$$Error = \frac{1}{2}\sum_{m=1}^{n_5}(d_m(T) - y_m(T))^2$$

where n_5 is the number of output dimensions in the GenSoFNN $d_m(T)$ and $y_m(T)$ denote the respective m^{th} desired and computed outputs based on the inputs evaluated at each training step T. The error signals and the updating rules start from layer 5 to layer 2 with respect to the training data vector pair $\{X(p), D(p)\}$, where $p \in \{1 \ldots P\}$ and P denotes the number of training patterns in one training epoch.

2.3.5 Example of Use of a GenSoYagerFNN with the XOR Dilemma

XOR dilemma is a classical non-linear problem used to illustrate the deficiencies of a single-perceptron network. It also shows the drawbacks of fuzzy neural networks that use the joint input-output partitions of the data space as fuzzy rules as well as clustering techniques that require the number of clusters to be predefined. It involves classification of 4 corners of a unit square in a two-dimensional space. A pair of diagonal corners is grouped as Class 0 while the remaining pair corners are grouped as Class 1 (Figure 2.11). We want to train the network if an initial training set of 4 points, described in the following file:

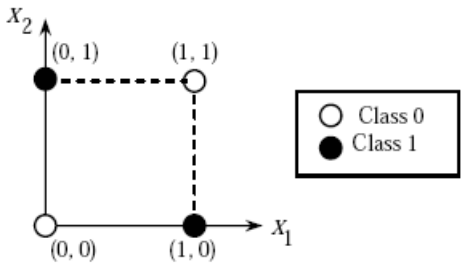

Figure 2.11: The XOR Dilemma [23]

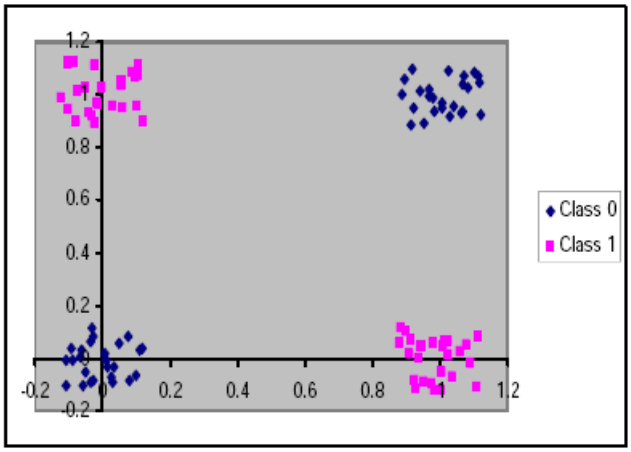

Figure 2.12: The XOR Dilemma results using the training test [3]

```
% Training data file for the XOR Dilemma
% x1 x2 output
  0  0  0
  0  1  1
  1  0  1
  1  1  0
```

With this simple training set, the network must be able to classify the different points. The test set consists of input data added with a rate of 20% of noise. Results are shown in figure 2.12.

Now let see how the GenSoFNN performs.
At the begin the network is composed with 2 inputs (x_1 and x_2 coordinates of the point) and one output (the class of the point). Then the data are fed into the network, and the result of the training gives 2 fuzzy sets for each input and 4 fuzzy rules. The fuzzy sets are the same for the 2 inputs, the first being the function describing the notion of "small" for a given input, and the second describing the notion of "large" (figure 2.13).

The four fuzzy rules crafted from the GenSoYagerFNN are the following:

Rule 1: if x_1 is small and x_2 is small then corner is class 0

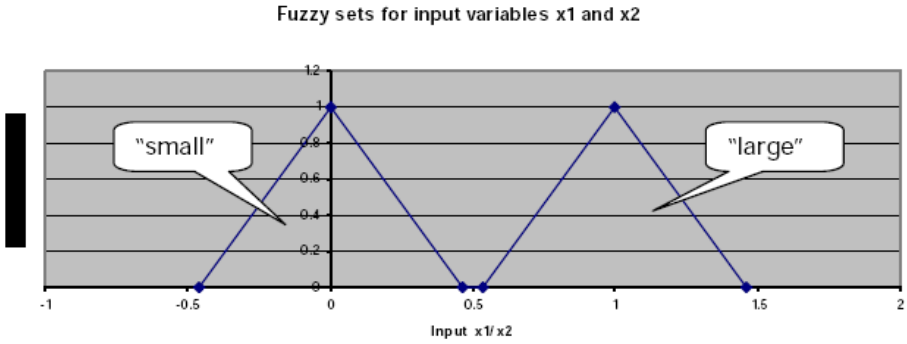

Figure 2.13: Fuzzy Sets for the XOR Dilemma [3]

Rule 2: if x_1 is small and x_2 is large then corner is class 1
Rule 3: if x_1 is large and x_2 is small then corner is class 1
Rule 4: if x_1 is large and x_2 is large then corner is class 0

We can see that the network is very transparent and easy to understand compared to neural networks, but has the same learning capabilities.

Chapter 3

The Driving Simulator (TORCS)

3.1 The Simulator

The simulator used in the C^2i has been developed and improved by Toh Mary et al. and has been involved in many projects [29] [30] [31]. The aim of this simulator is to provide a realistic environment to experiment on automated vehicles, using data from a human driver, and then learning with a fuzzy-neural network (figure 3.1).

The main problem with this simulator is that the driving physics, like the centripetal force that causes the car to slip from its turning course during high speed cornering, aren't well implemented. Or it's from the physics model that we can obtain accurate experimental results to justify the reliability of the Gen-SoFNN to handle the driving of a real car.
The implementation of the physics would have take too many time in order to be realistic, so we decided to change of simulator.

Between all the available simulators, two retained our intention: TORCS [1] and Racer [2]. Racer is the most professional one, with really good graphics and physic engine. But TORCS has some advantages that may argue for the final

[1] http://torcs.sourceforge.net
[2] http://www.racer.nl

Figure 3.1: The C^2i car simulator [32]

Figure 3.2: Screenshot of TORCS

choice. First it is open-source, so we can use it under the GNU Licence, and all the future versions will still be available. Second it's code source is really modular, which can help us when we want to add some features, and different students are working on at the same time. TORCS also allow to easily construct new robot drivers (it has been developped to compete robots of developpers), and has a software which permit to construct tracks without computer graphics knowledge (contrary to Racer). For all these reasons, we decided to work with the TORCS simulator.

Description of the TORCS simulator

> *TORCS (Figure 3.2), The Open Racing Car Simulator, is a car racing simulation, which allows you to drive in races against opponents simulated by the computer. You can also develop your own computer-controlled driver(also called a robot) in C or C++. TORCS is "Open Source" (GNU General Public License Version 2 or later).*
>
> *There are 42 different cars, 30 tracks and more than 50 opponents to race against. You can steer with a joystick or steering wheel, if it's supported by your platform. It is also possible to drive with the mouse or the keyboard, but it's not easy. Graphic features lighting, smoke, skidmarks and glowing brake disks. The simulation features a simple damage model, collisions, tire and wheel properties (springs, dampers, stiffness, ...), aerodynamics (ground effect, spoilers, ...) and much more. The gameplay allows different types of races from the simple practice session up to the championship. Enjoy racing against your friends in the split screen mode with up to four human players.*

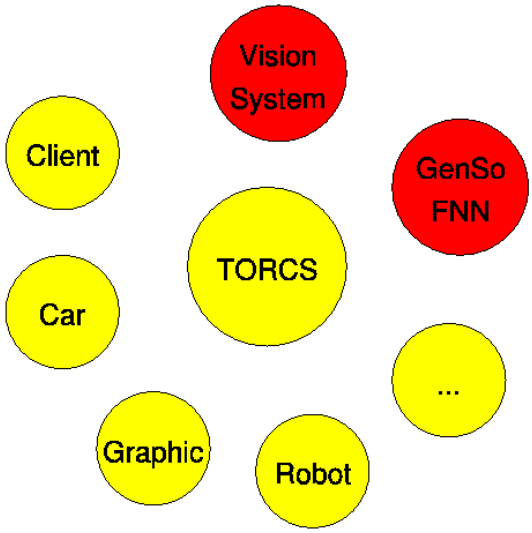

Figure 3.3: TORCS Modules

The use of the TORCS simulator helped us to have quickly a working simulator for our project. The car physics was fundamental for testing the speed adaptation in the curves (in the case of the past simulator, we didn't have to slow down when we were in curves, so the simulator couldn't correctly learn how to adapt its speed). The improvement of graphics also added realism to simulations, which is important when we use a vision system (via a camera) in order to collect data from the environment.

3.2 Developping modules

As we said earlier, TORCS source code is based on a very modular approach (Figure 3.3). So it's easy to add new modules, according to our needs.
The first modules we had to add were the GenSoYagerFNN (modified version of the GenSoYager library developped by Richard Oentaryo under Windows [3]), and the vision system. The vision system module is still under development by another student.
Developping the libraries under modules will allow to update TORCS without changing anything, and for students to work on different projects without conflicting each other.

To test our experiments, we need a driver. TORCS is also made to easily create new autonomous drivers (robots), which are created like modules and loaded in shared library. Testing drivers is consequently an easy step, where you just have to implement algorithms in functions of the template robot. We can create as many drivers as we want.

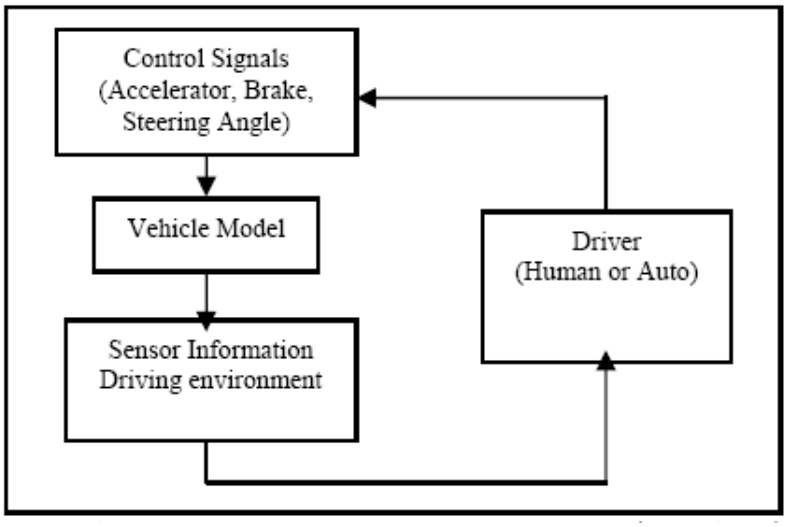

Figure 3.4: Vehicle driving control sequence [3]

3.3 The Car Model

The car model incorporates four control signals: *steering, brake, accelerator and gear*. The gear can be controlled either in manual or automatic mode.
To drive manually, we just have to configure a new player (in the configuration panel of the TORCS menu), with the controls used (keyboard, mouse, racing wheel ...) and the gear mode (automatic or manual). The driver can then be loaded like the other robots (human driver is then a special robot).
In manual driving, the human driver applies control signals based on its perception about the environment and its driving aptitude. The resulting human-driving data are then logged and used to train the fuzzy-neural network which serves as controllers in auto-driving mode. The trained network applies the three control signals in order to drive the car.
The driving agent can be modeled as a state machine whose output depends on the state of the simulated vehicle. The control signals are input into the vehicle model. Subsequently computations are done and the simulation environment and sensors will be changed and updated. Information of the driving environment is then feedback to the driver agent. The driving sequence can be illustrated by figure 3.4.

3.4 Training Data Collection

The simulator is used to collect the training data that capture different driving maneuvers in a given road scenario. The drivers use the visual feedback to decide the subsequent action to be taken. For instance, if the driver saw a turn ahead, he/she might respond by turning the steering wheel in a certain direction. Figure 3.5 concludes the training data collection of human driving performance. The log file stores all the actions taken including the sensor information of each simulation time interval. The information is then used to train

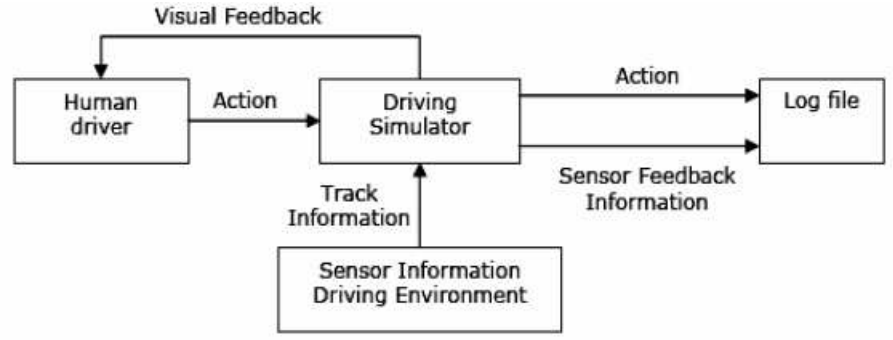

Figure 3.5: Training data collection [3]

the neuro-fuzzy system within the simulation.

There's one log file per network to train (so three log files: one for the steering system, one for the throttle system and one for the brake system). A log file is composed with a first line containing the number of training data, the number of inputs and the number of outputs, the next lines being the inputs and outputs at each time step. Inputs used are described in the next chapter, the output of the steering system being the angle of the wheel, and the output of the throttle system being the position of the throttle pedal.

Once the human training set recorded, we can use it to train the network. If the training set is coherent, the system should be able to automatically drive the car, reproducing the human driving, but also being able to manage new situations, as we do when we learn to drive: a monitor shows us how to drive on a specific road, but then we can generalize what we learned and drive on every road.

Chapter 4

Realization of Intelligent Speed Adaptation in TORCS

4.1 Example of Using ISA with the Active Acceleration Pedal

The Active accelerator pedal (see figure 4.1) provides a counter-force whenever the driver tries to depress it beyond a pre-set speed limit [33]. The performance of the vehicle is not affected at speed levels below the pre-set maximal speed. The Active accelerator pedal also restricts the engine's fuel injection when the vehicle reaches the actual speed limit.
Knowledge on the factors influencing driver speed-behavior, the relationship between speed and traffic safety and means to influence speed behavior is systematized. Expert judgments and the high proportion of injury accidents show that improved speed adaptation has the largest safety potential in the following situations:

- Road surface, visibility and weather affected situations (e.g. slippery road, fog, darkness).
- Places where sudden speed reduction is needed (e.g. motorway exits, sharp bends).
- Encounters with crossing-course both between motor vehicles and between motor vehicles and pedestrians/bicyclists (e.g. at intersections, at zebra crossings).

In this project (like in all others projects encountered in the domain of ISA) curves aren't taken into account. But drivers can easily be caught out by sharped curves, especially in campaign roads.
Designing a system which can adapt its speed according to the degree of a curve is an innovative research, and can as well be used for automated vehicles as for helping drivers by automatically decreasing the speed, or warning them.

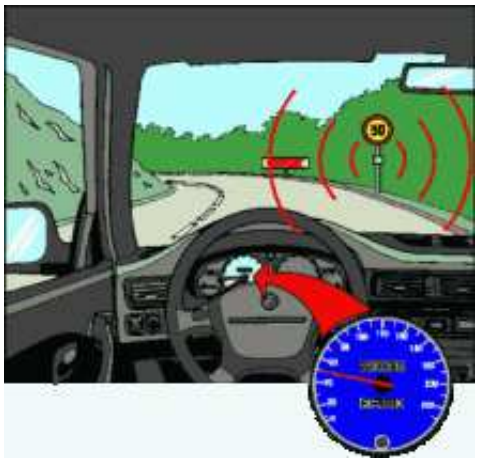

Figure 4.1: The active acceleration pedal interacts with the road [a]

[a]http://www.tft.lth.se/research/ISA.htm

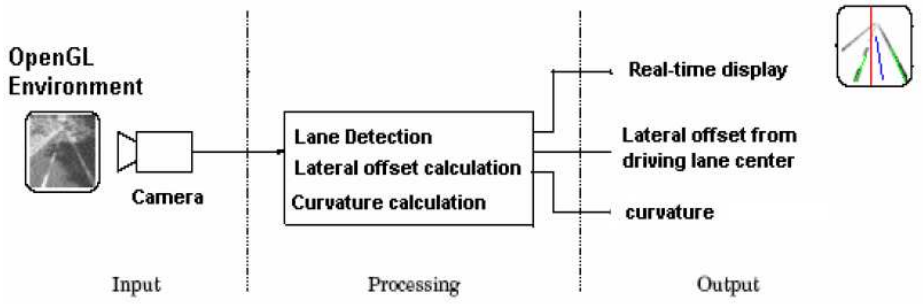

Figure 4.2: Block diagram of vision system (adapted from [32])

Controlling the speed can only be down with understanding how humans do. Some psychological studies have been made about the control of both lateral and longitudinal controls on humans [34] [35], and about what the human watch when steering [36] [37] [38].

Through the next sections we will see how to implement intelligent speed adaptation, by processing the vision of the environment via a camera in input, and sending processed information to a fuzzy neural network.

4.2 The Vision System, a Method for Lane Following

4.2.1 An Overview of the Vision System

The vision system we use in our project is a modified version of the Wu Guowei's [32]. Figure 4.2 shows the block diagram of the vision system, specifying the input and the output of the vision system.

The images are captured via a camera positioned inside the vehicle. The

lane detection system detects the different lane markings in the road images, using the Sobel edge detection algorithm.
After lane detection, we have different useful information, such as the vehicle heading direction and its position between the lane marking. With such information, we can calculate the *lateral offset* of the vehicle from the driving lane center (distance between the vehicle center from the approximated lane center). The *degree of curvature* of the road can also be computed by calculating the difference between the lane center in front of the vehicle, and a farther-off lane center.

The lateral offset was already implemented. We had to adapt the vision system to TORCS, by first modifying it to make it proper work under Linux, and then adapt to extract information from the buffer of TORCS. The vision of the two simulator being different, modifications were necessary on the vision system to extract good information.
We also had to implement the curvature calculation.

4.2.2 Lane Detection

The lane detection is made by the Sobel edge detection applied to the image. Once edges detected, we apply Hough transform to detect the lane the car's on. Finally, the lateral of set and the degree of curvature are calculated upon these preceding results.

Sobel Edge Detection

Images edges are defined as local variations of image intensity. The gradient magnitude $e(x, y)$ can be used as an edge detector:

$$e(x, y) = \sqrt{f_x^2(x, y) + f_y^2(x, y)}$$

where $f_x(x, y)$ is the x component of the gradient vector, and $f_y(x, y)$ is the y component of the gradient vector.
Alternatively, the sum of absolute values of partial derivatives f_x, f_y can be used for faster computation:

$$e(x, y) = |f_x(x, y)| + |f_y(x, y)|$$

Local edge direction, $\theta(x, y)$, is given by:

$$\theta(x, y) = arctan(\frac{f_y}{f_x})$$

The Sobel edge detection provides good performances and is relatively insensitive to noise [39] (see figure 4.3 for an application of the Sobel edge detection on an image). Sobel edge detection uses two convolution kernels, one to detect changes in vertical contrast (K_x), and the other to detect horizontal contrast (K_y):

$$\mathbf{K_x} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad \mathbf{K_y} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

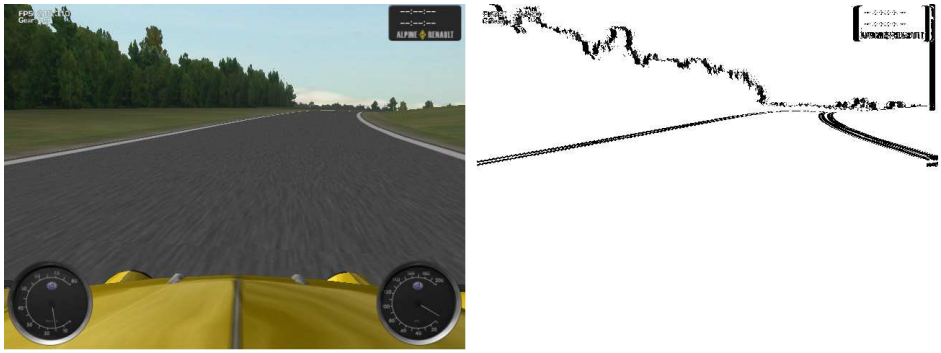

Figure 4.3: Sobel edge detection

The algorithm used for detecting edges and highlighting the lane markers is as follows:

1. `Convolve image` $I(x, y)$ `with` $K(j, k)$ `to get` $R(x, y)$
2. `Mark peaks in` $||R(x, y)||$ `that are above threshold` T

Applying convolution K to image I can be represented as:

$$R(x, y) = \sum_{k=-1}^{l} \sum_{k=-1}^{l} K(j, k) \cdot I(x - j, y - k)$$

The threshold operation is given as:

$$R(x, y) = \begin{cases} 1 & R(x, y) \geq T \\ 0 & \text{otherwise} \end{cases}$$

where T is experimentally determined to be 0.5.

Hough Transform

The Hough transform is a technique which can be used to isolate features of a particular shape within an image [40]. Because it requires the desired features to be specified in some parametric form, the *classical* Hough transform is most commonly used for the detection of regular curves such as lines, circles, ellipses ... The main advantage of the Hough transform technique is that it is tolerant of gaps in feature boundary descriptions and is relatively unaffected by image noise.

The Hough transform map the lines in image space to a point in parameter space. The axes of Hough space are the line parameters θ and ρ, based on the parametrization:

$$x \cdot cos\ \theta + y \cdot sin\ \theta = \rho$$

where x, y are the x and y coordinates of a point on the line, and θ is the angle from origin.

Each line is represented by a single point (a', b') in the parameter space (a, b). From figure 4.4, a straight line passes from two image points $(x_1, y_1))$

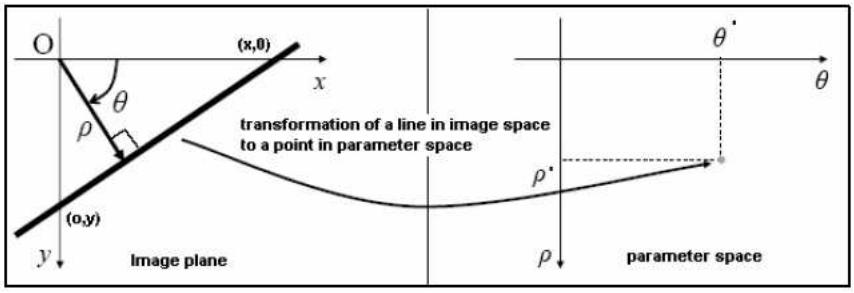

Figure 4.4: Representation of straight line in image plane and its equivalent in parameter space [32]

and (x_2, y_2) on the image plane. Any line passing through the point (x_1, y_1) corresponds to the line $b = -a \cdot x_1 + y_1$ in the parameter space. Similarly any line passing through (x_2, y_2) corresponds to the line $b = -a \cdot x_2 + y_2$ in the space (a, b). The intersection (a', b') of these two lines uniquely determines the straight line through both (x_1, y_1) and (x_2, y_2).
The procedure for straight line detection is as follows:

1. The parameter space is discretized and the parameter matrix $P(a, b), a_1 \leq a \leq a_k, b_1 \leq b \leq b_l$ is formed.
2. For every pixel (x_i, y_i) that posses value 1 at the binary edge detector output, the equation $b = -a \cdot x_i + y_i$ is formed.
3. For every parameter value $a, a_1 \leq a \leq a_k$, the corresponding parameter b is calculated and the appropriate parameter matrix element $P(a, b)$ is increased by 1.

The parameter matrix $P(a, b)$ shows the number of binary edge detector output pixels that satisfy equation above. If the number is above a certain threshold the line is declared and drawn.

Lateral Offset

The lateral offset is calculated as follow: the lane markers are approximated by Hough transform (green segments in figure 4.5). The car center (red line) indicates the direction and position of the car. The distance between the nearest point of the virtual lane center (blue line) and the car center is the *lateral offset*. The vehicle aims to keep this distance to a minimum.

Curvature Calculation

The first version of the implementation of the curvature calculation we made is very simple: once image processing (by applying Sobel edge detection then Hough transform) is made, we draw horizontal lines. The first is just at the front at the vehicle, and is for the actual position of the car with regard to its current lane (this is roughly the lateral offset of the car).
The second line is situated approximately in 10 meters of the first line (roughly

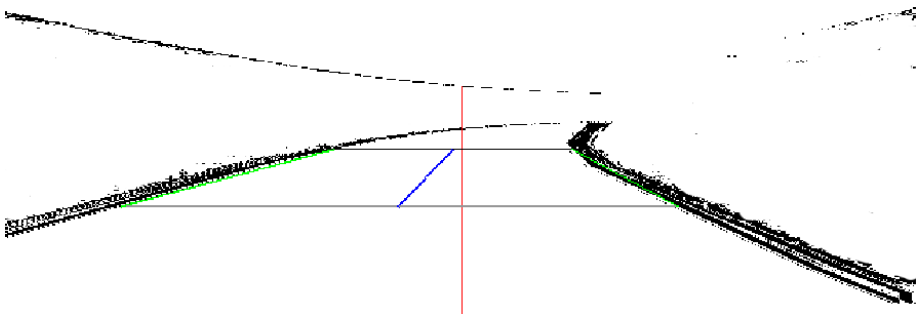

Figure 4.5: Lateral offset and angle curvature calculation

15 meters of the car). This will allow us to calculate the curvature for a maximum of 15 meters of the vehicle (we will see later how this could be improved).

In order to calculate the angle of curvature, we take the segment of these two lines, cut by the result of the Hough transform (so we have the segments positioned in the lane of the vehicle). Then we take the middle point of these segments (represented by the blue line on figure 4.5), and we calculate the angle of curvature as follow:

$$\theta = arctan(\frac{x_2 - x_1}{y_2 - y_1})$$

with θ the curvature of the road and $(x_1, y_1), (x_2, y_2)$ the lower and upper points of the virtual lane center.

4.2.3 Alternative Approaches to the Actual Vision System

This lane detection system is not the most optimized but is sufficient for our preliminary tests. Better systems can be found in literature, since a lot of lane detection and lane tracking systems have been implemented and have proved their efficiency. For references on different lane detection systems, one can look at [41] [42] [43] [44].
A better lane detection system could allow us to see farther away for computing the curvature of the road, so we could have 2 parameters, one for the curve the car will be engaged on now, and one for the curve which will be engaged on later (which is necessary for anticipating the curve). Actually our system can see a curve at 15 meters far away, which is not enough compared to the human vision.

4.3 Study of Human Visual Feedback

A lot of experiments have been made in order to understand where the human look, and what processed information is acquired, when he drives a car. Most of these reports come from a psychological branch, but the results can be applied for the robot driving. Knowing what the human perceive is a plus, and can help us to choose the inputs the networks need.

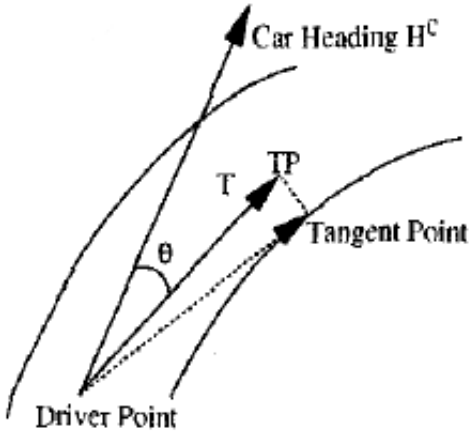

Figure 4.6: Definition of the tangent point [47]

[45] [46] [47] [48] made experiments on the direction the drivers look while negotiating curves. Results show that the driver often look at the tangent point on the inside of each curve (see figure 4.6). The main information processed by the eye are the distance of the tangent point (represented by T), and the direction of the tangent point, regarding the car's heading (represented by θ).

[37] [49] [34] rather concentrated their research on retinal flow. Retinal flow is more used when no lane is marked to help the driver (like driving in a car park, field or forest, or even driving in the night [50]). Although these experiments are very interesting, they are not helpful for our problem, but it might be useful to keep them in mind to improve our system while driving in unusual tracks or car parks.

Recent researches have also been made on how humans control their speed while steering [35]. This article is the human approach of our problem, and though is a good help for understanding how to slow down when approaching a curve. An algorithm is given for computing the good speed, given a curve and the distance to this curve, but this can only be useful without adding anticipation to our model (though the algorithm can only calculate the optimal speed for the current moment/position).

The study of human driving helped us to choose the good variables to feed the networks. We are now going to see how we have implemented the lateral and longitudinal control.

4.4 Lateral Control

Lateral control has been a big focus of research about autonomous vehicles [51]. A lot of models have been proposed, most of them based on sensori-motor control loop algorithms. Although these models work well on simple roads (like highways, without difficult negotiation of curves, or road-crossing), a few are

Figure 4.7: The steering subsystem scheme

able to operate on every kind of road. unsurprisingly, most of these models use the same inputs as humans.

Our model use the GenSoYagerFNN to compute the steering output, according to its inputs. The inputs we use can easily be computed from the vision system (i.e. camera input), by good algorithms commercially available. The inputs we will use are the lateral offset and the angle of curvature (with a look-ahead of 5 meters, see figure 4.7). Our model is not perfect, and still need improvement, for example by adding anticipation.

A typical simulation process is done as follow:
The driver will drive several laps on a track, the data are recorded every 40ms (data recorded are the lateral offset, the curvature angle, and the steering angle, which corresponds to the inputs and output data of our neural network).
Then we feed the network with the collected data (the structure and parameters of the networks are learned with the backpropagation algorithm), and then the robot might be able to drive on the track. The ability of the robot to drive widely depend of the choice of its inputs, but also of the way they are normalized. The normalization we decided to take after a lot of experiments are the following:

lateral offset The lateral offset is normalized between 0 and 1 in this manner (width is the width of the current track, and lat the lateral offset):

$$output = \begin{cases} 0.5 & \text{if lateral offset} = 0 \\ \frac{1}{2 \cdot e^{\frac{lat}{width/2}}} & \text{if lateral offset} < 0 \\ 1 - \frac{1}{2 \cdot e^{\frac{lat}{width/2}}} & \text{if lateral offset} > 0 \end{cases}$$

This normalization give more sensitive values when the lateral offset is near 0 (i.e. when the driver is near the middle of the lane), until the driver leave the lane (the value is then around 0.8), and then rises to 1 when the lateral offset rises to infinity.

curvature angle First, we normalized the angle between $-\Pi/2$ and $\Pi/2$, then if the angle is greater than $\Pi/2$ (or smaller than $-\Pi/2$), then $angle = \Pi/2$ (or $angle = -\Pi/2$). Basically, this mean that if the curve 5 meters in front of the vehicle is sharper than 90 degree, we can't turn no more harder, so we don't need to normalize more than between these values
Then we use this formula to normalize the angle between 0 and 1: $\frac{\sin angle + 1}{2}$.
This means that the values are more sensitive while the angle is small (as we more often encounter small curves).

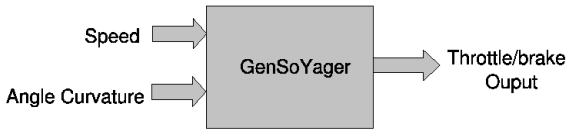

Figure 4.8: The throttle/brake subsystem scheme

Though these values are the best found during our experiments, the car is only able to turn on simple tracks. Therefore we need to create a more robust lateral control system, by adding anticipation (for example).

4.5 Longitudinal Control

In order to implement the longitudinal control, we use two other networks, one for the throttle gas, and another for the brake gas. The input use for the both networks are the same: the speed of the vehicle and the angle curvature (see figure 4.8).
The normalization of the angle curvature is the same than in the lateral control, and the normalization of the speed is simply a linearization between 0 and 200km/h.

The results obtained were good, but didn't include an anticipation of curves (i.e. the car was just slowing down at the last moment, when it saw the curve). Or humans know that a curve will soon arrive (by seeing it in advance, or through a sign), and slow down at the good distance (most of times). That's why we need to include anticipation, through a variable, described in next section.

4.6 Adding Anticipation to the Longitudinal Control

Controlling the throttle and brake pedals isn't made in a reflex way, for the humans. We first analyze the road and begin to decelerate before arriving in the curve. Consequently, a simple sensori-motor control can't be applied if we want to secure the longitudinal control, and anticipate the upcoming curves.
Our idea was to calculate anticipation in a variable, which will be fed to the GenSoYagerFNN. Then, the GenSoYager needs to variables, one is for the anticipation of the upcoming curve, and the other is the actual speed.

Two inputs are used for the anticipation, the speed and a new variable, adding the anticipation to our system (see figure 4.9). We will now describe how to anticipate the curve.

Every 40ms, we calculate the angle of the curve at a certain distance of the car (the distance depends of the speed, see table 4.1).
The returned value is then added to an array (in FIFO mode), of 40 values. The array contains the description of the curves of the track for 1.6 seconds

Figure 4.9: The throttle/brake subsystem scheme, anticipation added

Table 4.1: The distance of the curve, depending of the speed

speed (in m/s)	**distance (in m)**
x < 4.7	5
4.7 < x < 7.8	10
7.8 < x < 10.9	15
10.9 < x < 14.1	20
14.1 < x < 17.2	25
17.2 < x < 20.3	30
20.3 < x < 23.4	35
23.4 < x < 26.6	40
26.6 < x < 29.7	45
29.7 < x < 32.8	50
32.8 < x < 35.9	55
x > 35.9	60

(40*40ms). This array will be used to calculate the anticipation of the curve. We take the values of every 4 steps in the array, to have 10 different values. we then calculate the derivation of the values, and add them to a global result:

```
for(int i = 3; i < 39;i+=4) {
  if(curvature[i] > 0 && curvature[i+4] > 0)
    res += (curvature[i+4]-curvature[i])/0.16;
  else if(curvature[i] < 0 && curvature[i+4] < 0)
    res += (curvature[i]-curvature[i+4])/0.16;
  else if(curvature[i] < 0 && curvature[i+4] > 0) {
    if(-curvature[i] > curvature[i+4])
      res += (curvature[i]-curvature[i+4])/0.16;
    else
      res += (curvature[i+4]-curvature[i])/0.16;
  }
  else if(curvature[i] > 0 && curvature[i+4] < 0) {
    if(curvature[i] > -curvature[i+4])
      res += (curvature[i+4]-curvature[i])/0.16;
    else
      res += (curvature[i]-curvature[i+4])/0.16;
  }
}
```

With the algorithm we used we can see if the curve is soft or sharp (the bigger

the result is, the sharpest is the curve), and also if the we are at the beginning or at the end of the curve. If we are at the beginning the curve become more and more sharp, and then the result will be positive, but if we are at the end the curve become more and more soft, and the result will be negative.

In result, the vehicle will not react as soon as it sees a curve, but when a limit will be attained (the variable *res* become bigger and bigger while we approach a curve). So the variable depends from the sharpness of the curve, but also from the time we will arrive in the curve. It can be see as a sort of memory the robot has of the shape of the road for the next 1.5 seconds.

The data we use for the time step or the memory duration are taken from the paper [52] where experiments were made on human drivers. They found that an occlusion of 1.5 seconds on human drivers was the maximum limit before resulting a performance degradation. Therefore, it is the duration of memory the robot has. Some experiments can be made to see if a longer or shorter memory can affect the capacity of anticipation of the robot.
They also that action taken by human was changed every 40ms (when changed were needed), and it is the time step we use to process image and take action.

This algorithm is useful for anticipating the curves, and give us good results, as we will see in the next chapter.

Chapter 5

Experimental Results and Analysis

5.1 Training of the GenSoYager

In order to make the autonomous driving work, we had to collect data from a human driver, and make the network learn from these collected data.
For the moment just one simple track is presented to the driver (other tracks will soon be implemented to generalize the tests), see figure 5.1.

The track has two long straight lines to accelerate, and two curves to test the intelligent speed adaptation. the beginning of the curves are soft and then become harder, so we shall see the car slowly decelerate and it arrives in the curve, and then more decelerate once it engaged in the curve.

So we asked the driver to drive one lap, with accelerating the maximum on the straight lines, and to decelerate progressively when he see the curve approaching, keeping a good but security speed.

5.1.1 Training the Steering Subsystem

Our first work was to reproduce the previous results on our new simulator. So we wanted the car to be able to correctly steer in the curves. In order to reproduce these tests, we just saved the lateral offset (for the input) and the steering angle (for the output), to train the GenSoYager. The driver had to keep the car on the middle of the lane while driving. A constant speed was given (slow enough not to feel the lateral forces while driving in curves) to concentrate only on the lateral control. After training the network with the human data, the system was correctly driving the car, driving smoothly even in curves.the results obtained were the same that the last student work, and the interpretation of these results can be found in [32].

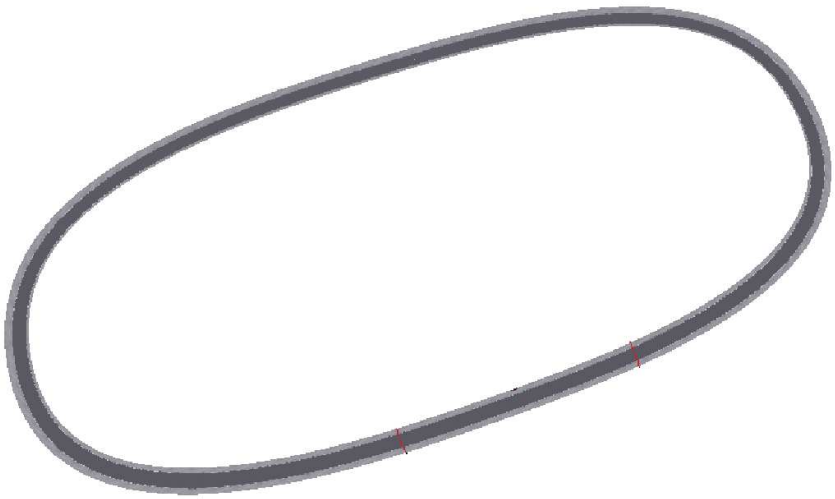

Figure 5.1: The test track

5.1.2 Training the Throttle Subsystem

Then, we had to train the throttle subsystem for the system to correctly adapt its speed in curves.
For this test, the driver had to concentrate on the lateral and longitudinal control. The way a human control the steering and throttle when he drive can be found in [35]. For the lateral control, we just saved the same data than in the last section, and for the longitudinal control, we saved the angle curvature and the speed of the vehicle (for the input), and the throttle (for the output). Our system had now to couple the longitudinal control, with the previous results of the lateral control.

For the training of the GenSoYager, we used the parameters described here:

Input/Output Threshold	0.5
Input/Output STEP	0.1
Input/Output SLOPE	0.6
Input/Output Annex Threshold	0.8
Max Training Epoch	50
Target Error	0.00005
Learning Constant	0.005

Once we trained the network with the given training set, the structure of the GenSoFNN was as shown as follow:

Parameters	**Throttle System**
Input Labels	9
Output Labels	4
Input label breakdown per input dimension	[2,7]
Number of Rules	12

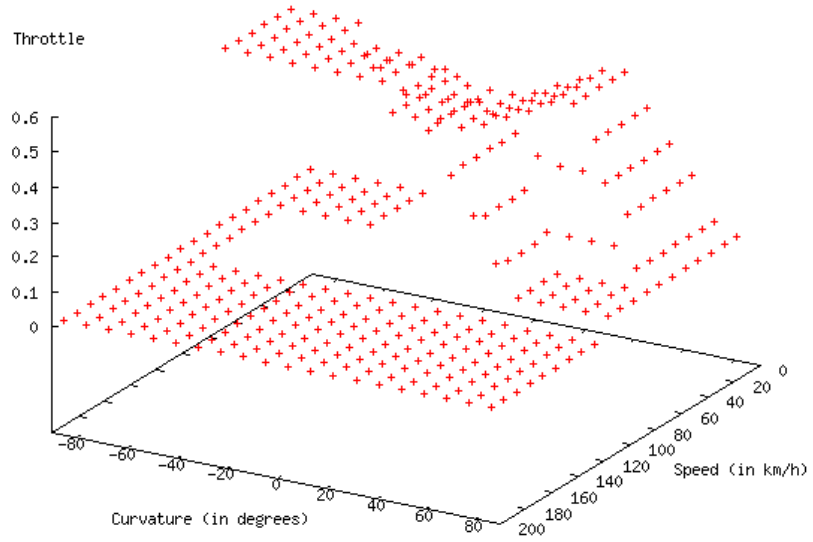

Figure 5.2: The throttle output, given speed and angle curvature in inputs

As we can see, 2 fuzzy labels were formed for the vehicle speed, and 7 for the angle of curvature and a total of 12 rules were formed.
Given this structure, the system was correctly driving the car, adapting its speed before and in curves not to drive too fast to go off road.

We must now analyze the defined network to understand how the system is able to drive the car.

5.2 Analysis of the Throttle Subsystem

Let now see how the car accelerate or decelerate, given the current speed and the curvature angle (see figure 5.2).

The speed vary between 0 and 200 km/h, and the curvature angle between -90 and 90 degrees (from left to right). If the speed input is upon 100km (the maximum speed the human driver was on the straight lines), the car totally stop accelerating. Otherwise, we can see that the car accelerate and decelerate according to the speed of the vehicle, but also to the degree of curve of the road. When the current speed is quite slow (between 0 and 50 km/h), the car slow down only if the road has a big curve (between 50 and 90 degrees) and the more curved is the road the more the car slow down.
Between 50 and 80 km/h, the progressively slow down for a smaller degree of curve, and then for 80 to 100 km/h, the car begin to slow down if the curve is

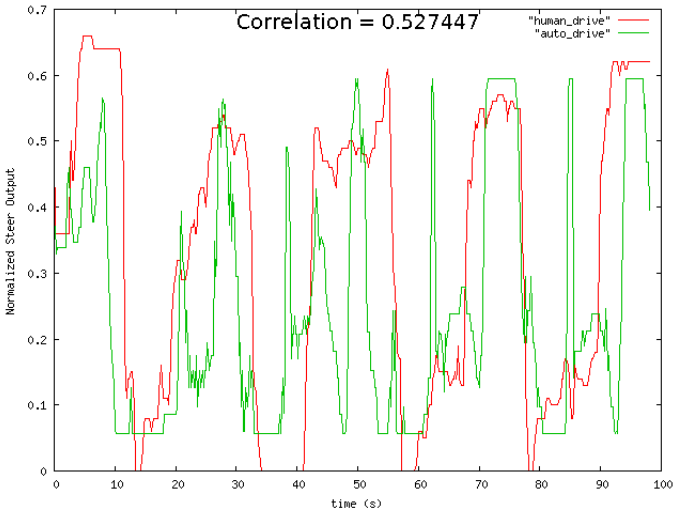

Figure 5.3: Pearson's correlation between human and auto driving

more than 20 degree.
With these results, we can see that the car can correctly and smoothly adapt its speed on the track. Coupled with the lateral (i.e. steering) control, this give a automated driving control, which can drive like humans, accelerating on straight lines, and slowing down in curves, to prevent going off road.

We can also see that the car should not be able to adapt its speed for left curves (on the graph, the car accelerate until a curve of 60 degrees, and then totally stops accelerating). This is due to the fact that in our track, there is no left curve, so our car only learned to drive with right curves.

We will now analyze the differences between the human driver and the auto-driver.

5.2.1 Recall Results

To evaluate the network recall ability, the *Pearson product-moment correlation coefficient* [53] was used.

Pearson product-moment correlation coefficient:
The Pearson's correlation coefficient between two vectors $X = x_1, \ldots, x_n$ and $Y = y_1, \ldots, y_n$ with mean $\overline{x}$ and $\overline{y}$ respectively is defined as:

$$Correl(X,Y) = \frac{\Sigma(x-\overline{x})\cdot(y-\overline{y})}{\sqrt{\Sigma(x-\overline{x})^2\cdot\Sigma(y-\overline{y})^2}} \qquad \text{where } Correl(X,Y) \in [-1;+1]$$

The square error of the output by the auto driver was also calculated.

We wanted to calculate the correlation between the throttle output of the human and the auto-driver. The result gave us 0.527447, which means that

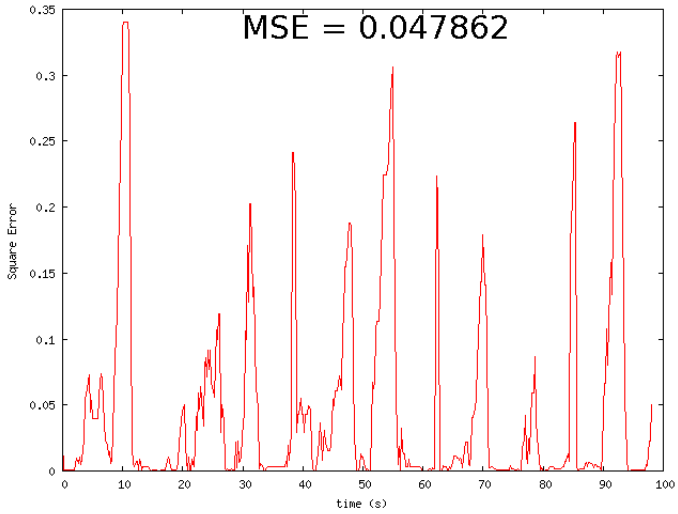

Figure 5.4: Square Error between human and auto driving

the auto-driver approximately accelerate and slow down like the human driver, but there are some differences. We can see on the figure that in general, the auto-driver accelerates before the driver, and decelerate after. This mean that the auto-driver is faster than the human for taking decisions (i.e. accelerating after a curve), but is also less preventive (i.e. slowing down when arriving in a curve). These results are normal, we saw that the implemented vision system couldn't see as far as the human vision system, so the car see the curve after the human, and so decelerate after. This is not really a problem, as the car adapt wisely its speed, and doesn't arrive too fast in curves. The fact that the car accelerate before humans can be explain that humans don't react instantly to what they see (on the curve we can see a difference of 2 seconds between the re-acceleration of the auto-driver and the human).

The mean square error is relatively low (4,8%, see figure 5.4) which means that the auto-driver usually drives at the same speed than the human driver. The differences of speed are usually during the beginning of phases of accelerating or decelerating maneuvers, or when the auto-driver smooths the throttling (human throttling is not really precise, this is due to the fact that for the moment the throttle and steering control is made with the mouse, which is harder to control than a wheel and pedal; a wheel joystick will be soon implemented to have a better control of the vehicle).

5.3 Rule Firing Analysis

A study of the rules fired in the network was conducted to analyze the network ability to maintain the consistency of the rule base. A concise and consistent rule base is significant to ensure good performance and a critical factor when applied to an embedded control system with limited storage capacity. A consistent rule base is shown by comparing the proportion of rules actually used

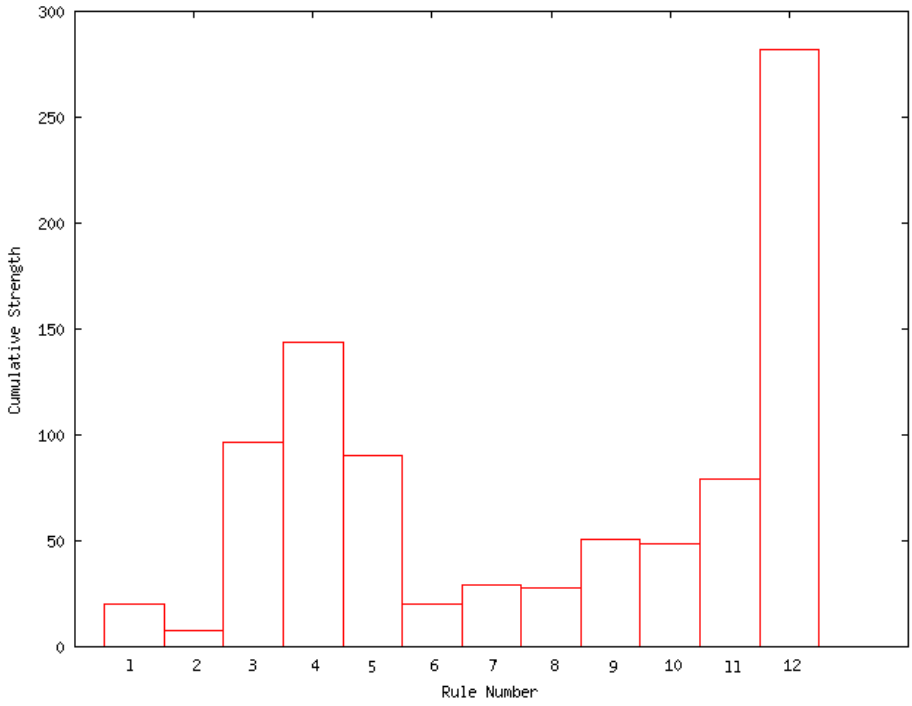

Figure 5.5: Rule Firing Strength

compared to the total number of rules in the network. A consistent rule base is best displayed by a wide spread of the rules being fired over the total number of rules for all possible driving situations.
The experimental results for the rule firing strength throughout the experiment process are summarized in figure 5.5.
We can see that the rule base generated by the GenSoYager is consistent, as all the rules are used (the main rule being the number 12, and rules 3, 4, 5 and 11 being the auxiliary rules).

To illustrate the intuitiveness and the ease of interpretation of the fuzzy rules crafted by the GenSoYager, we will now analyze the rule base of our throttle system.
There are a total of 12 rules, linked to 9 input labels and 4 output labels. The inputs are two features: the speed of the vehicle and the degree of curvature of the road. The speed of the vehicle has 2 fuzzy sets, and the curvature has 7 fuzzy sets. The throttle output has 4 fuzzy sets (see figure 5.6 for the structure of the network).

The fuzzy sets of the antecedent layer described the different fuzzy values the inputs can have (see figure 5.7), we can attach linguistic terms to these values :

- input "Speed" = {slow=1, fast=2}
- input "Angle Curvature" = {straight=3, soft left=4, right=5, soft right=6, hard right=7,extreme right=8, left=9}

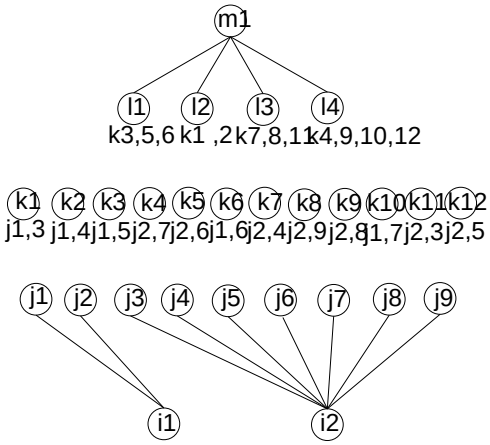

Figure 5.6: The GenSoYager structure for the Throttle System

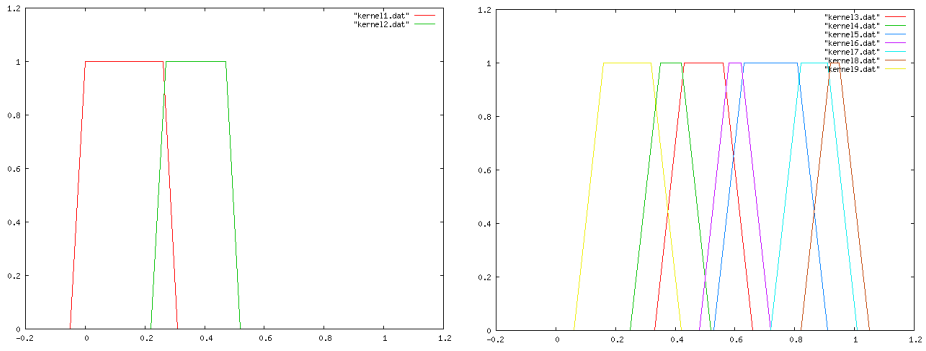

Figure 5.7: Labels of the two inputs

The number following the linguistic terms is the number of the neuron interpreting this label.
For the consequent layers, only the centroid of the kernel is needed, which is given by the following table:

Neuron	Centroid
1	0,46
2	0,33
3	0,59
4	0,06

This values can be interpreted as follow:

- output “Throttle” = {normal acceleration=1, slow acceleration=2, hard acceleration=3, no acceleration=4}

The corresponding fuzzy rule extracted from the rule base of the throttle system can be formulated with the linguistic terms as follow:

1. **IF** *speed* is *slow* **AND** *curvature* is *straight* **THEN** *throttle* is *slow accel*

2. **IF** *speed* is *slow* **AND** *curvature* is *soft left* **THEN** *throttle* is *slow accel*
3. **IF** *speed* is *slow* **AND** *curvature* is *right* **THEN** *throttle* is *normal accel*
4. **IF** *speed* is *fast* **AND** *curvature* is *hard right* **THEN** *throttle* is *no accel*
5. **IF** *speed* is *fast* **AND** *curvature* is *soft right* **THEN** *throttle* is *normal accel*
6. **IF** *speed* is *slow* **AND** *curvature* is *soft right* **THEN** *throttle* is *normal accel*
7. **IF** *speed* is *fast* **AND** *curvature* is *soft left* **THEN** *throttle* is *fast accel*
8. **IF** *speed* is *fast* **AND** *curvature* is *left* **THEN** *throttle* is *fast accel*
9. **IF** *speed* is *fast* **AND** *curvature* is *extreme right* **THEN** *throttle* is *no accel*
10. **IF** *speed* is *slow* **AND** *curvature* is *hard right* **THEN** *throttle* is *no accel*
11. **IF** *fast* is *slow* **AND** *curvature* is *straight* **THEN** *throttle* is *fast accel*
12. **IF** *speed* is *fast* **AND** *curvature* is *right* **THEN** *throttle* is *no accel*

We can see that the rule base is coherent with the human way of driving, apart for the left turning (which is normal, as the system didn't learned how to turn left). We can also see that if the car speed is slow, and the road is straight, acceleration is slow. This is due to the fact, that the car speed is slow on a straight line only at the start (then the car speed is always fast on straight lines). So when we start driving, we go slow, and then increase our speed (like in real life).

Comparing the rules with the rule firing strength, we can see that the first rule is not often used (like we just say, just at the start). The 3 left turning rules are also not often used, as for the slow and soft right rule (we don't arrive slowly on a soft curve!).
The most used rule is the number 12, which means that we often try to go fast on right curves, when they are not too hard (this rule is the most often used as we more often drive on the right curves than on the straight lines).

The above illustration shows that the fuzzy rules are intuitive to human cognitive process. It's also very easy to extract the rules of the fuzzy neural network, and so to understand it, compared to the "black box" of the neural networks.

These results are very encouraging for the continuation of our project, but we still have a lot of work to do, these are just a preliminary step for having a real working intelligent adaptive speed system.

5.4 Tests for the anticipation control

During the second part of our project, we tried to add an anticipatory system to our longitudinal control.

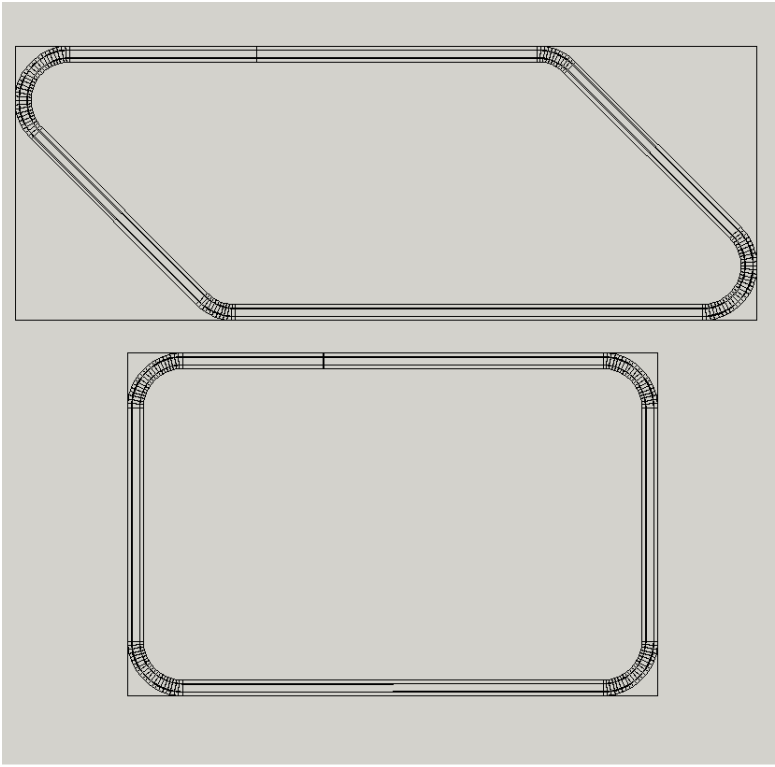

Figure 5.8: exemple of tracks

First we had to create some tracks for testing on simple circuits. 12 tracks were created, with a different curve angle for each (6 for left curves and 6 for right curves, see figure 5.8).

The driver has to drive for 3 turns, and we record the data every 40ms. He drives on curves of 45, 90 and 135 degrees, and of different radius. Then the autonomous driver must be able to interpolate and extrapolate the results.
We compare the results with the human by calculating the lateral offset MSE (explain), the time to do a tour, the maximum speed, and the anticipation of the curve.

5.4.1 First test: driving on the 90 degree curves

So we first drive 3 turns on this track, and then train the steering, throttle and braking networks with the recorded data. The results obtained are described in the table 5.1.

We see that the robot driver can drive faster than the human driver, without exceeding the limit (130 km/h). The driver exceeds it, particulary when he is concentrated on the road, and doesn't watch the speedometer. The human anticipation was also really differing in each curve (varying between 22 to 70 meters), but the robot anticipation is always around 45 meters before the curve.

Table 5.1: comparaision between the human and robot driver

	human driver	**robot driver**
MSE	1.18	2.93
TIME (in sec)	169	164
MAX SPEED (in km/h)	152	128
Anticipation (mean in m)	38	45

The only driving skill the robot driver was not as good as the human driver was for the lateral driving (the robot was more driving near the left lane than in the middle of the lanes), this is shown by the MSE which is higher for the robot. A better system must be found for the steering network, in order to drive more properly.

In conclusion, we can see that the robot drive in a safter way than the human driver. It doesn't exceed the road limit, slow down graduatly quite in advance and has a smoothly acceleration/deceleration.

5.4.2 Second test: driving on the 45 degree curves

For this second test, we will drive on a track with 8 curves of the same degree (45 degrees), but with different radius (of 50,100,150 and 200 meters).
The goal of this test is to see if the robot driver reacts differently according to the radius of the curve.

We first record the data of the human driver, and then train the network with these one. We look how long the driver begins to slow down before arriving in the curve. The results are shown above:

Table 5.2: Slow down performance, according to the radius

Radius	**next curve (in m)**
200	0, 0
150	31, 34
100	44, 44
50	54, 52

We can see that the driver slows down according to the radius. This means that the sharper is the curve (i.e. the higher radius), the sooner the robot driver will begin to slow down.

The results found are the one we should expect for a human driver. Coupled with the last test, the results are very promising. The next step is to have a robust lateral control, to test the longitudinal control on every kind of road, to see if it can be generalized.

Chapter 6

Conclusion and Further Work

6.1 Conclusion

During this project, we have shown how to implement Intelligent Speed Adaptation in autonomous vehicles to adapt their speed when they arrive in curves, using the GenSoYagerFNN. The fuzzy neural network possesses the learning technique of neural network to detect hidden relation in driving data, and the capabilities of fuzzy rule based to handle inherent uncertainty of the driving environment and reason approximately. The GenSoYagerFNN was able to craft a rule base of human driving expertise and perform the same driving behaviors.

We show that by training training the GenSoYagerFNN, the system was able to correctly adapt its speed according to the type of curve. The system was not able to anticipate the curves as a human do, but this was due to a lack of robustness in the construction of the vision system. New test must be done with a better vision system to see if the car can anticipate curve like humans. A lot of good performing lane tracking systems are actually available, so we can get one already constructed instead of implementing our.

The Intelligent Speed Adaptation is a quite new research area, especially for adapting speed on curves, so the results are promising, but there is still a lot of work to do in this domain. The first thing to do is to generalize the results, for the car to be able to drive in any road, and then in any situation (with other cars on the road, several lanes, vision obstruction...)

The implementation of a "memory" to the system could help us to manage these complex system (for example, we can remember that there's a curve, and adapt our speed, even if the curve is hidden by a car in front of us).

Figure 6.1: photo of the Cycab vehicle [a]

[a] http://www.inrialpes.fr/iramr/pub/Orccad/Presentation/intro-eng.html

6.2 Further Work

For the continuation of our project, a lot of different work can be made, the first one is improving the system to be more robust.
During the next weeks, we will try to generalize the results, for the car to be able to drive and adapt its speed on every track. We must also find how to anticipate the curves like human do. If these results are rapidly found, we can change the orientation of our project in two distinct ways.

First, We can try to adapt evolutionary algorithms to the system, in order to make it learn how to drive without human help. Evolutionary algorithms in robotic systems often give better results than other techniques [54]. this is due to the fact that learning from human data means to also learn human errors, and evolutionary computation is a fast and lovely (inspired by the Darwin theory of the species evolutionary) method to find a good solution.

The last orientation that the project could take is to adapt autonomous driving on a real car. NTU own some cycab (see figure 6.1), so simple tests could be done on it.

Bibliography

[1] Sleet D. et al. Peden M., Scurfield R. World report on road traffic injury prevention. Geneva, 2004. World Health Organization.

[2] S.L. Jamson. *Intelligent speed adaptation: evaluating the possible effects of an innovative speed management system on driver behaviour and road safety.* PhD thesis, Institute for Transport Studies. University of Leeds. UK., 2001.

[3] R.J. Oentaryo. Automated driving based on self-organizing gensoyager neuro-fuzzy system, 2004.

[4]

[5] Michel Pasquier and Richard J. Oentaryo. Learning to drive the human way: A step towards intelligent vehicle. *International Journal of Vehicle Autonomous Systems, Special Issue on Integrated Vehicle Control: Theories, Methods and Experimentation*, to appear in 2007.

[6] Dean Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems 1.* Morgan Kaufmann, 1989.

[7] J. Forbes, T. Huang, K. Kanazawa, and S. Russell. The batmobile: Towards a bayesian automated taxi. In *Proc. of the 14th IJCAI*, pages 1878–1885, Montreal, Canada, 1995.

[8] Rahul Sukthankar, John Hancock, Shumeet Baluja, Dean Pomerleau, and Chuck Thorpe. Adaptive intelligent vehicle modules for tactical driving. In *Proceedings AAAI '96*, 1996.

[9] Rahul Sukthankar, Shumeet Baluja, and John Hancock. Evolving an intelligent vehicle for tactical reasoning in traffic. In *Proceedings of the International Conference on Robotics and Automation*, volume 1, pages 519 – 524, April 1997.

[10] Guarong Chen and Trung Tat Pham. *Introduction to Fuzzy Systems.* Chapman & Hall/CRC, 2006.

[11] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

[12] Kevin Gurney. *An Introduction to Neural Networks.* Taylor & Francis, Inc., Bristol, PA, USA, 1997.

[13] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. pages 15–27, 1988.

[14] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386Ž013408, 1958.

[15] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Netw.*, 2(5):359–366, 1989.

[16] Chin-Teng Lin and C. S. George Lee. *Neural fuzzy systems: a neuro-fuzzy synergism to intelligent systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[17] Detlef Nauck. A fuzzy perceptron as a generic model for neuro–fuzzy approaches. In *Proc. Fuzzy–Systeme'94*, Munich, 1994.

[18] B. Kosko. *Fuzzy Associative Memory Systems, Fuzzy Expert Systems*. Addison-Wesley, Reading, MA, 1986.

[19] N.R. Pal. E.K. Tsao, J.C. Bezdek. Fuzzy kohonen clustering networks. *Pattern Recognition*, 27(5):757–764, 1994.

[20] C.-C. Jou. A fuzzy cerebellar model articulation controller. *IEEE Internat. Conf. on Fuzzy Systems*, pages 1179–1186, 1992.

[21] R. W. Zhou and C. Quek. Popfnn: A pseudo outer-product based fuzzy neural network. *Neural Networks*, 9(9):1569–1581, Dec 1996.

[22] Quek Hiok Chai Ang, Kai Keng and Michel Pasquier. Popfnn-cri(s): A pseudo-outer product based fuzzy neural network using the compositional rule of inference and a singleton fuzzifier. In *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*.

[23] W.L. Tung and C. Quek. Gensofnn: a generic self-organizing fuzzy neural network. *IEEE Transactions on Neural Networks*, 13(5):1075–1086, 2002.

[24] E. H. Mamdani. Application of fuzzy logic to approximate reasoning using linguistic synthesis. In *Proceedings of the sixth international symposium on Multiple-valued logic*, pages 196–202, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[25] W.L. Tung. A generalized platform for fuzzy neural network. Technical Report ISL-TR-01/01, School Comput. Eng, Nanyang Technol. Univ., Singapore, 2001.

[26] L. A. Zadeh. Calculus of fuzzy restrictions. pages 210–237, 1996.

[27] James M. Keller, Ronald R. Yager, and Hossein Tahani. Neural network implementation of fuzzy logic. *Fuzzy Sets Syst.*, 45(1):1–12, 1992.

[28] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. pages 318–362, 1986.

[29] M. Pasquier, C. Quek, and M. Toh. Fuzzylot: a novel self-organising fuzzy-neural rule-based pilot system for automated vehicles. *Neural Networks*, 14(8):1099–1112, Oct 2001.

[30] K.K. Ang. *POPFNN-CRI(S): A Fuzzy Neural Network based on the compositional Rule of Inference*. PhD thesis, Nanyang Technological University, Singapore, 1998.

[31] R. J. Oentaryo and M. Pasquier. Self-trained automated parking system. In *Proceedings of the 8th IEEE International Conference on Control, Automation, Robotics and Vision (ICARCV 2004)*, 2004.

[32] J. Wu GuoWei. Fuzzy rule identification for an intelligent car in a dynamic highway system, 2005.

[33] Varhelyi A. Hjalmdahl M. Speed regulation by in-car active accelerator pedal effects on driver behaviour. *Transportation Research Part F.*, 7(2):77–94, 2003.

[34] John P. Wann and Richard M. Wilkie. How do we control high speed steering? pages 401–419, 2004.

[35] Wilkie R.M. and Wann J.P. Steering with an eye to braking.

[36] Brett R. Fajen, William H. Warren, Selim Temizer, and Leslie Pack Kaelbling. A dynamical model of visually-guided steering, obstacle avoidance, and route selection. *Int. J. Comput. Vision*, 54(1-3):13–34, 2003.

[37] J.P. Wann and D.K. Swapp. Why should look where you are going. *Nat Neurosci*, 3(7):647–648, 2000.

[38] A. Beall and J. Loomis. Visual control of steering without course information, 1996.

[39] Ioannis Pitas. *Digital image processing algorithms*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[40] P.V.C. Hough. Machine analysis of bubble chamber pictures. In *International Conference on High Energy Accelerators and Instrumentation*, 1959.

[41] K. Macek, B. Williams, S. Kolski, and R. Siegwart. A lane detection vision module for driver assistance. In *In Proceeding of the IEEE/APS Conference on Mechatronics and Robotics*, 2004.

[42] Yue Wang, Eam Khwang Teoh, and Dinggang Shen. Lane detection and tracking using b-snake. *Image Vision Comput.*, 22(4):269–280, 2004.

[43] Xiaofeng Hu Yong Zhou, Rong Xu and Qingtai Ye. A robust lane detection and tracking method based on computer vision. *Measurement Science and Technology*, 17(4):736–745, 2006.

[44] M. Bertozzi, A. Broggi, and A. Fascioli. Obstacle and lane detection on argo autonomous vehicle.

[45] M. F. Land and D. N. Lee. Where we look when we steer. *Nature*, 369:742–744, 1994.

[46] J.B.J. Riemersma. Attention and perception in negotiation of curves. In *Vision in Vehicles III*, Aachen, Germany, 1989.

[47] E.R. Boer. Tangent point oriented curve negotiation. In *Intelligent Vehicles Symposium, 1996., Proceedings of the 1996 IEEE*, Tokyo, Japan, 1996.

[48] M. F. Land. The visual control of steering. *Vision and Action*, pages 163–180, 1998.

[49] Wann J.P Wilkie R.M. Controlling steering and judging heading: Retinal flow, visual direction and extra-retinal information. *Journal of Experimental Psychology: Human Perception and Performance*, 29(2):363–378, 2003.

[50] Wann J.P Wilkie R.M. Driving as night falls: The contribution of retinal flow and visual direction to the control of steering. *Current Biology*, 12(23):2014–2017, 2002.

[51] C. Taylor, J. seck, R. Blasi, and J. Malik. A comparative study of vision-based lateral control strategies for autonomous highway driving, 1999.

[52] Boer ER Royden CS Hildreth EC, Beusmans JM. From vision to action: experiments and models of steering control during driving. *Journal of Experimental Psychology: Human Perception and Performance*, 26(3):1106–32, 2000.

[53] R.N. Goldman and J.S. Weinberg. *Statistics: An Introduction*. Prentice-Hall, 1985.

[54] S. Nolfi and D. Floreano. *Evolutionary Robotics. The Biology, Intelligence, and Technology of Self-organizing Machines*. MIT Press, Cambridge, MA, 2001. 2001 (2nd print), 2000 (1st print).